



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

USING OPEN SOURCE SOFTWARE IN VISUAL SIMULATION DEVELOPMENT

by

Ricardo Brigatto Salvatore

September 2005

Thesis Advisor:
Second Reader:

Dr. Rudolph Darken
CDR Joseph Sullivan, USN

Approved for public release; distribution is unlimited.

This thesis done in cooperation with the MOVES Institute

THIS PAGE INTENTIONALLY LEFT BLANK

| | | | | |
|--|---|--|--|--|
| REPORT DOCUMENTATION PAGE | | | Form Approved OMB No. 0704-0188 | |
| Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503. | | | | |
| 1. AGENCY USE ONLY (Leave blank) | | 2. REPORT DATE September 2005 | 3. REPORT TYPE AND DATES COVERED Master's Thesis | |
| 4. TITLE AND SUBTITLE: Using Open Source Software in Visual Simulation Development | | | 5. FUNDING NUMBERS | |
| 6. AUTHOR(S) Ricardo Brigatto Salvatore | | | | |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000 | | | 8. PERFORMING ORGANIZATION REPORT NUMBER | |
| 9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A | | | 10. SPONSORING/MONITORING AGENCY REPORT NUMBER | |
| 11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. | | | | |
| 12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited. | | | 12b. DISTRIBUTION CODE | |
| 13. ABSTRACT (maximum 200 words) <p>The convergence between personal computer-based games and virtual environments technologies dramatically reduced development costs and potentially increased the use of the technology in training activities. Using open source/free software tools in the process can expand these possibilities, resulting in even greater cost reduction and allowing the flexibility needed in a training environment.</p> <p>This thesis presents a configuration and architecture to be used when developing training visual simulations using both personal computers and open source tools.</p> <p>Aspects of the requirements needed in a visual simulation development, processes to develop an application and issues related to the use, licensing and selection of open source/free software are analyzed to identify their limitations and possibilities.</p> <p>This architecture was tested by developing a small visual simulation. The tools and engine used are presented to enable any future project applying open source software to follow similar procedures.</p> | | | | |
| 14. SUBJECT TERMS: Visual Simulation, Open Source Software, Game Development, Virtual Environment Training | | | 15. NUMBER OF PAGES 103 | |
| | | | 16. PRICE CODE | |
| 17. SECURITY CLASSIFICATION OF REPORT Unclassified | 18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified | 19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified | 20. LIMITATION OF ABSTRACT UL | |

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited.

**USING OPEN SOURCE SOFTWARE IN VISUAL SIMULATION
DEVELOPMENT**

Ricardo B. Salvatore
Lieutenant Commander, Brazilian Navy
B.S., Brazilian Naval Academy, 1989

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

**NAVAL POSTGRADUATE SCHOOL
September 2005**

Author: Ricardo Brigatto Salvatore

Approved by: Dr. Rudolph Darken
Thesis Advisor

CDR Joseph Sullivan, USN
Second Reader

Dr. Peter Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The convergence between personal computer-based games and virtual environments technologies dramatically reduced development costs and potentially increased the use of the technology in training activities. Using open source/free software tools in the process can expand these possibilities, resulting in even greater cost reduction and allowing the flexibility needed in a training environment.

This thesis presents a configuration and architecture to be used when developing training visual simulations using both personal computers and open source tools.

Aspects of the requirements needed in a visual simulation development processes to develop an application and issues related to the use, licensing and selection of open source/free software are analyzed to identify their limitations and possibilities.

This architecture was tested by developing a small visual simulation. The tools and engine used are presented to enable any future project applying open source software to follow similar procedures.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

| | | |
|------|---|----|
| I. | INTRODUCTION..... | 1 |
| A. | PROBLEM STATEMENT..... | 1 |
| B. | MOTIVATION..... | 2 |
| C. | APPROACH..... | 2 |
| D. | THESIS ORGANIZATION..... | 2 |
| II. | VISUAL SIMULATION TECHNOLOGY..... | 5 |
| A. | PERSONAL COMPUTERS BASED VISUAL SIMULATIONS | 5 |
| 1. | Introduction..... | 5 |
| 2. | Visual Simulation Definitions | 5 |
| 3. | Immersion, Interaction and Presence | 6 |
| 4. | Human Factors Software and Hardware..... | 7 |
| 5. | Vision and Displays..... | 8 |
| 6. | 3D Image Rendering Software | 9 |
| 7. | Hearing and Audio..... | 10 |
| 8. | Audio and Video Relation | 11 |
| 9. | 3D Audio Generation Software | 11 |
| B. | VISUAL SIMULATION TRAINING AND GAMES..... | 12 |
| 1. | Introduction..... | 12 |
| 2. | Motivation and Involvement..... | 13 |
| 3. | Training and Development Costs..... | 14 |
| 4. | Uses and Limitations..... | 16 |
| C. | VISUAL SIMULATION DEVELOPMENT | 17 |
| 1. | Introduction..... | 17 |
| 2. | Development Process | 17 |
| 3. | Authoring Tools and Engines..... | 22 |
| 4. | Development Problems..... | 23 |
| III. | OSS/FS SOFTWARE FOR VISUAL SIMULATION DEVELOPMENT | 27 |
| A. | OPEN SOURCE SOFTWARE | 27 |
| 1. | Introduction..... | 27 |
| 2. | Open Source Licensing..... | 28 |
| a. | <i>MIT / X License</i> | 29 |
| b. | <i>BSD License</i> | 29 |
| c. | <i>Apache License</i> | 30 |
| d. | <i>GNU General Public License (GPL)</i> | 30 |
| e. | <i>GNU Lesser General Public License (LGPL)</i> | 30 |
| f. | <i>Mozilla Public License (MPL)</i> | 31 |
| g. | <i>Q Public License (QPL)</i> | 31 |
| h. | <i>Artistic License (Perl)</i> | 31 |
| 3. | Problems of OSS/FS..... | 32 |
| 4. | Open Source Advantages | 34 |
| 5. | Hardware and Open Source Drivers Issues | 34 |

| | | |
|-----|--|----|
| B. | SELECTING AND EVALUATING OPEN SOURCE SOFTWARE | 35 |
| 1. | Introduction..... | 35 |
| 2. | Identification of Candidates..... | 35 |
| 3. | Reviews | 35 |
| 4. | Comparison..... | 36 |
| 5. | Analysis..... | 39 |
| C. | OPEN SOURCE TOOLS AND ENGINES FOR VISUAL SIMULATION DEVELOPMENT | 40 |
| 1. | Introduction..... | 40 |
| 2. | OpenGL | 42 |
| 3. | OpenAL..... | 43 |
| 4. | Delta3D Visual Simulation Engine..... | 43 |
| a. | <i>OpenSceneGraph.....</i> | 45 |
| b. | <i>Open Dynamics Engine (ODE).....</i> | 45 |
| c. | <i>Delta3D Viewer</i> | 45 |
| d. | <i>Delta3D Particle Editor</i> | 46 |
| e. | <i>STAGE (Simulation, Training and Game Editor)</i> | 47 |
| f. | <i>Other Libraries and Dependencies.....</i> | 47 |
| 5. | GIMP | 48 |
| 6. | Blender | 49 |
| 7. | Audacity | 49 |
| IV. | ANALYSIS, CONCLUSIONS AND FUTURE WORK | 51 |
| A. | ANALYSIS | 51 |
| B. | CONCLUSIONS | 52 |
| C. | FUTURE WORK..... | 53 |
| | APPENDIX A. INSTALLING AND SETTING LINUX AND DELTA 3D | 55 |
| | APPENDIX B. APPLICATION DEVELOPED USING OSS/FS TOOLS | 75 |
| | LIST OF REFERENCES..... | 81 |
| | INITIAL DISTRIBUTION LIST | 85 |

LIST OF FIGURES

| | | |
|------------|--|----|
| Figure 1. | FOV (From Ref. 7)..... | 9 |
| Figure 2. | Animation Production Pipeline (From Ref. 22). | 19 |
| Figure 3. | Game Development Hierarchy (From Ref. 17). | 20 |
| Figure 4. | Simplified Development Process Proposed..... | 21 |
| Figure 5. | Evaluation and Selection Process. | 40 |
| Figure 6. | Delta 3D Dependencies Architecture..... | 44 |
| Figure 7. | Delta 3D Viewer..... | 46 |
| Figure 8. | Delta 3D Particle Editor. | 47 |
| Figure 9. | Architecture Proposed. | 51 |
| Figure 10. | Fedora Core 3 Installation Type. | 56 |
| Figure 11. | Delta3D Package Downloaded..... | 58 |
| Figure 12. | Extract Delta3D Package. | 59 |
| Figure 13. | Delta3D File Tree Structure..... | 60 |
| Figure 14. | Show Hidden Files..... | 61 |
| Figure 15. | Old .bashrc File. | 62 |
| Figure 16. | New .bashrc File..... | 63 |
| Figure 17. | Extract Delta3D Lib Files. | 65 |
| Figure 18. | TestCharacter Screenshot..... | 67 |
| Figure 19. | TestEffects Screenshot. | 67 |
| Figure 20. | Viewer Screenshot. | 68 |
| Figure 21. | Desktop Icons Screenshot..... | 68 |
| Figure 22. | Create New File Desktop..... | 69 |
| Figure 23. | HelloWorld Screenshot..... | 72 |
| Figure 24. | Application Class Hierarchy..... | 77 |
| Figure 25. | The DDG Model..... | 78 |
| Figure 26. | DDG Turn and the Submarine in the Surface..... | 78 |
| Figure 27. | Underwater Camera with a Frigate View. | 79 |

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

| | | |
|----------|--|----|
| Table 1. | <i>America's Army</i> Costs (From Ref. 17) | 15 |
| Table 2. | General Software Licensing Costs | 23 |
| Table 3. | Open Source Licenses | 32 |
| Table 4. | Proprietary and Open Source Tools | 42 |
| Table 5. | Delta3D Dependencies Libraries List | 48 |

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

| | |
|--------|---|
| PC | Personal Computer |
| OSS/FS | Open Source/Free Software |
| 3D | Three Dimensions |
| 2D | Two Dimensions |
| API | Application Program Interface |
| GPU | Graphical Processor Units |
| COTS | Commercial Off the Shelf |
| DMSO | Defense Modeling and Simulation Office |
| VE | Virtual Environment |
| VS | Visual Simulation |
| HMD | Head Mount Display |
| OS | Operational System |
| OpenGL | Open Graphics Language |
| dB | Decibels |
| FOV | Field of View |
| DARPA | Defense Advanced Research Projects Agency |
| AA | <i>America's Army</i> Gamy Project |
| IDE | Integrated Development Environment |
| DOD | Department of Defense |
| MIT | Massachusetts Institute of Technology |
| FSF | Free Software Foundation |
| OSI | Open Source Initiative |
| GUI | Graphical User Interface |
| URL | Uniform Resource Locator |
| TCO | Total Cost of Ownership |
| SGI | Silicon Graphics |

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank my wife Silvia for her patience and courage in facing the challenge of living abroad with our two small kids. Thank you to my daughter and son for, while not understanding the reason why I was so busy, letting me have enough late working hours to finish this thesis.

Thanks to the Brazilian Navy for the opportunity to attend the Naval Postgraduate School and gain so much advanced knowledge.

Thanks to my advisor and the director of the MOVES Institute, Dr. Rudolph Darken, and to CDR Joseph Sullivan for their cooperation during the research process.

I would like to thank the International Office staff for the support they provided during my time in Monterey and in all other administrative issues.

I would like also to express a special thank you to the Delta3D project team, led by Erik Johnson, for their patience with my often naïve questions. Special thanks to Chris Osborne, who helped me when setting-up the Linux environment and shared his vast knowledge with me.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PROBLEM STATEMENT

During a visual simulation developing process, several support software tools are used. These include authoring tools for images, textures and three-dimension (3D) models, and engines, libraries and application program interfaces (API) to drive the video and audio rendering, and so on. These tools are normally high-cost proprietary software that has a tendency to lock the development from the very beginning, restricting the entire project from the start. These limitations dramatically increase the overall project dependency on specific vendors, and render development costs almost prohibitive for a training application. To use visual simulation technology for training, especially for massive military training, alternatives that allow freedom in development and customization at reasonable costs must be identified.

One possible way to achieve this freedom and to control costs is to use open source or free software (OSS/FS) tools and API in strategic parts of the development process. The successful introduction of OSS/FS in the development process could bring about an increase in the use of visual simulation technology in new areas where budget limitations are problematic. Questions addressed in this work are:

1. What are the possibilities and limitations of using game-like visual simulations in training?
2. What are the tools used during a visual simulation development?
3. How to select OSS/FS tools, libraries and engines?
4. Can these tools be used to develop visual simulations for training?
5. What is the process used when developing a visual simulation?
6. What are the advantages and disadvantages when using OSS/FS tools in visual simulations development?

B. MOTIVATION

The use of game-like visual simulations in several kinds of training is increasing daily as a generation of gamers joins the work market. Some of the known limitations inhibiting the use of visual simulations are vendor lock-in, lack of customization freedom and high development costs.

The main objective of this work is to explore possible technologies that exist in the OSS/FS world to allow a reduction in these limitations when developing visual simulations.

Another objective is to identify a process and OSS/FS alternatives for popular proprietary tools, with the goal of implementing a low-cost fast development process.

C. APPROACH

The first part of this thesis presents an overview of visual simulation technology, including human factors and the development cycle, focusing on personal computers (PC) hardware, and training applications.

The second part analyzes open source software in general, looking at licenses, OSS/FS evaluation and selection, existing tools and engines for visual and audio simulation rendering.

To illustrate the problems and difficulties encountered when using OSS/FS, an installation and setting tutorial for an open source visual simulation and game engine is described in Appendix A and, as a development test for the framework and tools proposed, a small visual simulation application development is described in Appendix B.

D. THESIS ORGANIZATION

This thesis is organized in the following chapters:

- I. Introduction
- II. Visual Simulation Technology
- III. Open Source Software and Visual Simulation Development
- IV. Analysis, Conclusions and Future Work

Appendix A: Installing and Setting Linux Fedora Core 3 and Delta 3D

Appendix B: A Visual Simulation Application Developed Using OSS/FS

THIS PAGE INTENTIONALLY LEFT BLANK

II. VISUAL SIMULATION TECHNOLOGY

A. PERSONAL COMPUTERS BASED VISUAL SIMULATIONS

1. Introduction

As outlined in Gordon E. Moore's 1965 Moore's Law¹, the CPU microchip evolution has reached now the Graphical Processors Units (GPU), making a commercial off the shelf (COTS) computer more powerful than the old specialized workstations used to run virtual environments and traditional simulators. Visual Simulation applications normally confined to those environments migrated to regular PCs and, today, a commercial computer game has almost the same visual impact as a real simulator.

In this chapter, we focus on the use of a PC to host a 3D Visual Simulation, evaluating human factors and some hardware and software issues. The emphasis is on open source software and regular commercial-off-the-shelf (COTS) hardware. OSS/FS is detailed in Chapter III.

2. Visual Simulation Definitions

Because confusion exists when referring to names and definitions, some clarification is provided below. Names such as Visual Simulation, Virtual Environment, Virtual Reality, Artificial Reality, Synthetic Environment, Virtual Worlds, etc., can all refer to the same thing. Some formal definitions are:

- For the Defense Modeling and Simulation Office (DMSO), a Virtual Environment (VE) is an environment that does not exist in the real world, and uses a computer-generated three-dimensional environment to create the immersion effect².
- According to J. K. Caird, Visual Simulation is a 3-D graphic image that is generated by computers to create a cognitive and physical interaction³.
- Furness and Barfield say that a Virtual Environment is a representation of a computer model or database that can be interactively experienced and manipulated by participants⁴.

There are a few common points in these definitions, namely, that computer generated images must exist and these images must allow interaction in a way that results in a kind of immersion.

3. Immersion, Interaction and Presence

Interaction can be considered the cognitive and physical real time participation of the user in the environment. It is the effects that the user causes in the environment and the reaction that he notices from the environment. It is an act-reaction relationship. According to the Merriam-Webster Dictionary is mutual, or reciprocal, influence.

For Slater, immersion is the sense of “being there.” It is the extension in which the virtual reality becomes dominant over the real world. It is when users, after experiencing the virtual environment, remember the experience as having visited a “place” rather than just having seen a database of images generated by a computer⁵.

But the ability to measure the sense of presence is low; the most commonly accepted theory is that as this sense increases, the user starts to experience the immersion effect. A VE must give the user the sensation that what he is virtually experiencing is almost real. Empirically, from the gamer’s experience, the idea that a PC can create a VE sufficiently immersive can be inferred. It is when the player plays a computer game for hours, completely forgetting the exterior world.

Definitions of visual simulations or virtual environments and the idea of immersion lead to three types of VE, each distinguished by the level of immersion achieved:

- Fully immersive, when the VE encompass the world in 360 degrees.
- Augmentations (Overlay) to the real world, when a transparent, or semi-transparent screen is used to project images on top of real world image, or the real world itself.

- Less-immersive, when we have “Through the Window” worlds, the viewer is positioned outside the VE.

Using a modern COTS PC, some kind of immersion can be achieved if certain hardware and software points are adjusted. First, the tasks desired to be performed in the VE must be analyzed. The expected result from the VE activity must also be analyzed.

Virtual Environments are widespread and used in several areas, such as entertainment, design and development, training, education, scientific visualization, collaboration, medicine and marketing. Each area has different requirements, and for many of them, if not all, the main motivation to use PC Virtual Environments is cost reduction.

Of course there will be a trade-off between cost reduction and immersion, especially on the hardware side. A rigorous task analysis can define what is expected, and determine what can be achieved. A PC solution probably will not achieve 100% of the tasks, but it could be a more realistic solution depending on the budget involved. Thinking in a solution that follows the 80/20 Pareto principle, 80% of the budget can be concentrated on the 20% most important characteristics to get immersion.

For example, for a fully immersive VE, expensive hardware like Head Mount Displays (HMD) and special input devices (gloves) are needed. The software licenses and development to manage these devices are costly and the increase in the immersion effect could not be significant enough to justify the cost.

4. Human Factors Software and Hardware

The interaction between a human and the environment, real or virtual, occurs through the senses. It is therefore important to understand a little about how the human sensation system works in order to analyze how a Virtual World stimulates these senses.

A VE is supposed to generate multimodal stimulation in human senses. According to Heilig, the estimated attention humans get from the environment

using which sense is⁶: vision (~70%), hearing (~20%), smell (~5%), touch (~4%) and taste (~1%). But these percentages depend on the age of the sample (population), profession, cultural differences and other factors. The important point is that vision and hearing are the predominant senses.

The actual PC hardware and software can simulate both sound and vision, with more or less realism, depending basically on the use of specialized input/output devices which reflects direct in hardware costs.

5. Vision and Displays

Two aspects related to human vision and displays are important: the frame rate and the field of view. The regular frame rate, or refresh rate of the displays used in a PC, must be above the human eyes' flicker fusion point, which is around 50-60 Hz. With modern monitors, achieving this rate is not a problem.

The Field of View (FOV) is more limited by the PC hardware because of the size of the monitors. FOV is the visual angle subtended at the viewer's eye. The natural FOV in humans are 120° vertical and 180° horizontal. Eye Station Point (ESP) is the physical location of the viewer's eye, the point where the viewer is located in relation to the display or monitor. Eye Station Point Geometric (ESPg) is the point in space that corresponds to the visual angle of the model as projected out of the display; it is the represented model center of projection determined by the virtual lens programmed in the software used to render the model image, and not by the monitor itself. Field of View Geometric (FOVg) is the visual angle of the model, not the display, subtended at the computer's virtual eye⁷.

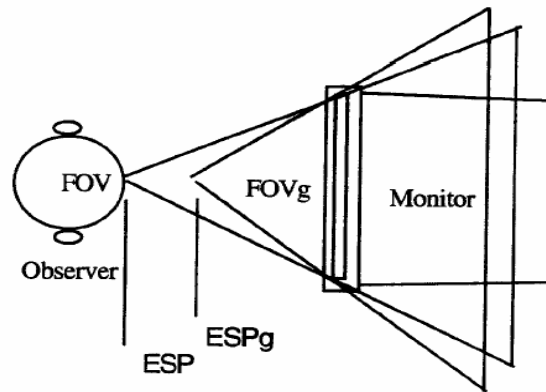


Figure 1. FOV (From Ref. 7).

Findings from the Psotka, Lewis and King study ⁷ suggest that when centered in the user, a 60 degree FOVg is optimum and robust across a range of tasks in perspective displays adopting regular viewing locations (from 350 mm to 900 mm) from the display.

A 17 inch regular PC monitor and a user within the standard viewer location (55 cm) can reach an FOV of 80° horizontal and 45° vertical. This is very reasonable and allows for a 60° model FOVg. A certain level of immersion can be obtained depending on the external influences. Better results can be achieved if the size of the monitor is increased or more than one monitor is used.

Using Head Mount Displays, a much more immersive VE can be created, but, in this case, the costs are high and they are not yet considered regular hardware.

6. 3D Image Rendering Software

On the software side, rendering 3D images in a regular PC display is not difficult anymore. Every video game renders 3D images using one kind of technology or another. What is needed is a good graphics card adapter that supports one of the main technologies: DirectX or OpenGL.

These are the two dominant technologies and both are incorporated in mainstream video card hardware. Microsoft DirectX, runs only on Microsoft Windows operational system (OS) and Open Graphics Language (OpenGL), is an open standard created by SGI that runs on several OSs.

Choosing the open source solution for a low-level 3D graphics API led us to several other open source options for higher-level 3D graphics, which are discussed in Chapter III.

7. Hearing and Audio

Sound is the other important feature for increasing the immersion effect; previous studies show that 3D spatial sound helps localization and memory retention in a virtual space.

The sounds humans can hear are described in terms of sound intensity and frequency. Humans can hear sounds in frequencies between 20 and 22,000 Hz. The intensity of sounds encountered in an everyday experience are in the range of 80 to 90 decibels (dB). The pain limit is around 120 dB. Normally, sound systems use 16 bits to represent the acoustic signal that provides a dynamic range of 90 dB, which is commonly considered sufficient for most simulations⁸.

As previously mentioned, audio stimulus covers a substantial percentage of the human senses. Technologies like surround sound, digital surround sound, stereo sound, and so on, are available for use in regular PCs. They can create a very reasonable immersive auditory experience by virtually distributing the sound sources in the space.

On the hardware side, developers must choose between using headphones or speakers. It is easier to suppress the real world sound and create spatialized sound using headphones. Using this option, full control of the amount of sound received by each ear can be achieved. Speakers can create spatial sound in an entire room, but the room itself has to be acoustically prepared and must be able to deal with the cross-influence between the two ears and shadow areas, which makes the task more difficult and costly.

To create a spatial sound effect, it is necessary to “virtualize” the sound source(s), i.e., create virtual sound sources around the space. This is accomplished via two factors: controlling the direction from which sound is coming by physically positioning speakers and reflectors, and managing the perceived distance by introducing delays and managing intensity, which is done using software.

8. Audio and Video Relation

One issue that cannot be overlooked is the synchronicity between sound and video: unsynchronized sounds and images can generate an uncomfortable sensation that reduces the immersion effect and can produce simulator sickness.

In their study, “Computational Requirements and Synchronization Issues for Virtual Acoustic Displays,” Miner and Claudell⁹ showed that humans can easily perceive mismatches in audio/visual cues. They found that it is necessary to keep the delays between audio and video as low as 100 ms to reduce this effect. The sound system used has to generate at least delayed directional sound waves allowing the creation of 3D sound cues.

The use of open field techniques (loudspeakers) is more appropriate for applications where the 3D space localization is not as important and ambient sound is enough. They are typically less expensive, as it is easy to find COTS sets of speakers in several configurations. The closed field technique (headphones) is better when a deployable solution is needed.

Regular PC sound cards can deliver several kinds of 3D sound solutions, and the techniques depend on the manufacturer. The two most widely used are A3D and EAX. Both are supported for almost all 3D sound card hardware. The high level sound API normally supports both of them.

9. 3D Audio Generation Software

The sound engines API offers are much more limited than the image rendering engines API, but there are still several solutions available. For example, in Windows there is Direct Sound, and in the open source world there is OpenAL.

Direct Sound and Direct Sound 3D are the two most common low-level sound APIs. They are attached to Microsoft Direct X technology, which is the reason they are almost a standard for 3D sound generation in the windows world.

OpenAL is an open source and cross-platform API that is becoming well supported by the majority of hardware vendors, can run in open source OS and is compatible with A3D and EAX sound cards. It is a natural choice for the open source world. OpenAL is discussed later in Chapter III.

Having analyzed the human factors of PC-based simulations, the next chapter discusses the training use of this technology.

B. VISUAL SIMULATION TRAINING AND GAMES

1. Introduction

One of the first uses of PC visual simulations was for entertainment in applications such as flight simulators and first person shooter games. Because of their success and improvement, these applications came under consideration as possible training solutions.

The use of entertainment devices for training is not new. In World War II the need for a fast-training cycle brought the flight simulator technology, an entertainment toy, to the military training world. This evolution was motivated by the possibility of offering low-cost training in safe conditions before engaging pilots in real flight. After the PC boom in the early 1990s, simulations were brought to the general public and the computer game industry started to apply the same software technologies used in military simulators to computer games. The success of the computer game industry made the game technology surpass the visual simulation technology, applying more realistic video and sound. According to Dr. Michael Zyda, citing the 1997 "Report from the Computer Science and Telecommunications Board of the National Research Council,"¹¹ games and interactive entertainment, not defense research expenditures, have become the main technology drivers for virtual environments. Modifications of COTS computer games are being applied in military and other kinds of training as well. For example, according to the DARWARS project website:

DARWARS is a DARPA-funded project to accelerate the development and deployment of the next generation of experiential training systems. These low-cost, web-centric, simulation-based systems take advantage of the ubiquity of the PC and of new technologies, including multi-player games, virtual worlds, intelligent agents, and on-line communities.¹²

There are some characteristics of games and visual simulations that are interesting to note: a game has a goal and competition, and is winnable and emotionally involving. It is also fun (or it is supposed to be fun) for the player, has no obligatory relation with reality and creates challenges for the player¹⁰. Simulations, on the other hand, are realistic, as physically correct as possible, involve no competition, and are methods for learning.

Despite these differences, they have a lot in common and we can go from simulations to games easily by introducing some game characteristics into traditional simulations.

The use of games in training is an attempt to increase training transfer and retention, increase trainee motivation and involvement, reduce costs, and maintain a flexible and agile training cycle¹⁰.

2. Motivation and Involvement

Motivation and Involvement are key points to be analyzed when trying to increase training transfer. One way to achieve these points is to emotionally involve the trainee in the process. This is a technique the entertainment and game industry uses very successfully.

This new generation of trainees is known by several names, such as the Wired Generation, the Connected Generation, and the Digitally Born Generation. Dr. Michael Macedonia stated that this generation established different thinking processes, characterized by multiprocessed thinking (doing more than one thing at the same time and fast spanning attention between tasks); concentrating focus on multimedia and not pure text; learning experience preference changing from passive to experimental learning; reasoning is more concrete than abstract and deductive; thoughts are organized in a more easily accessible way (like

databases), rather than in a linear step by step way; and the ability to work on tasks over a substantial period of time with full concentration if well stimulated¹³.

This echoes research conducted by the Jupiter Research Company for advertising agencies¹⁴. The favorite leisure activities for men between 18 and 34 years old are web surfing (22%), watching TV (22%), playing videogames (16%), watching movies (15%), reading a book or magazine (6%) and listening to or playing music (4%).

Prensky says that people growing up in the last 20 years are digital natives and have spent 10,000 hours playing video games, read 250,000 emails, talked 10,000 hours on cell phones, watched 20,000 hours of TV, 500,000 hours of commercials and spent less than 5,000 hours reading a book¹⁵.

These are the kinds of people that need to be motivated to learn or train. The training technology to be used must fit them. In military training, it is a real challenge to adapt a training technology to get these people well trained in an efficient way in the shortest time possible.

Several examples can be noted as initiatives to integrate games, PC simulations and military training: the previously mentioned DARWARS project¹², the creation of the Institute of Creative Technologies (ICT) by the Strategy Training and Instrumental Command (STRICOM) of the U.S. Army and the University of Southern California¹⁶, the Army Game project started at the MOVES Institute at the Naval Postgraduate School, which created the game *America's Army*, an advertising game designed to help increase Army recruiting.

3. Training and Development Costs

It is clear that the use of PC hardware is cheaper than the use of big simulators or real live training. Visual Simulations running on inexpensive hardware can have a very fast software development cycle, much like game development, when a game modification can be developed in weeks and run on almost any existing PC.

The important point is to concentrate efforts on the most important part of the simulation, audio and video, and to analyze whether the training transfer is

satisfactory by mapping the training tasks according to what can be done using live training, big simulators and PC simulators.

However, even game-like development is not cheap enough to be applied in massive military training. The game industry is mainly driven by high cost proprietary development tools and products. Even though cheaper than a complete full simulator development, a custom game development still has high costs for limited budget scenarios.

From the *America's Army* (AA) development experience, it is important to note that the development cost could be as high as four million dollars. The main costs involved are personnel payment and software licensing. In AA, a team of about twenty-six people was used and the game-engine licensing (UNREAL) had a large influence on the final cost¹⁷.

| Typical Costs | Year1 | Year2 | Year3 | Year4 |
|-----------------------|---------|--------|--------|--------|
| Game Engine and Tools | \$ 300K | \$100K | \$100K | \$400K |
| Development Costs | \$2.0M | \$2.5M | \$2.5M | \$2.5M |
| Operational Costs | \$1.5M | \$1.5M | \$1.5M | \$1.5M |
| Total | \$3.8M | \$4.1M | \$4.1M | \$4.4M |

Table 1. *America's Army* Costs (From Ref. 17)

Game-engine costs include the engine itself and authoring tools. Game engine licensing is a problem in using this model for massive training. All modern game or visual simulation software development is based on the engine, a computer graphics API, which will be discussed later. If it is necessary to distribute the source code for customizations, license price can be a problem, as stated in the paper, "From Viz-Sim to VR to Games: How We Built a Hit Game-Based Simulation."¹⁷

Traditional simulators, in addition to software development and licensing, require specialized hardware for motion or mockups to reproduce a real scenario. Prices increase in all areas where these features are added and flexibility is lost. For example, to develop and install a simulator for a Landing Craft Air Cushioned Vehicle (LCAC) was estimated to cost twenty-nine million dollars in 2000, not including annual maintenance and operational costs. The price of bringing people to a centralized location for training must also be included¹⁸ in operational costs.

4. Uses and Limitations

Games and PC simulations cannot be a substitute for every kind of simulator or real training. Three dimensional simulation games, however, allow users to engage in cognitive training, think process training.

As graphics technology advanced, PC applications started to be used also for familiarization training, mission rehearsal and hypotheses evaluation, and, when the networks were added, they started to be used also in team coordination and basic tactics training¹⁹.

However, there are limitations when using this kind of training. For example, to simulate people when a behavioral-based situation is needed, limited choices render the simulation unrealistic. When trying to mimic interpersonal situations, like a conversation, some artificiality is also introduced and the thinking process will be different from a real scenario. When simulating very dynamic systems, like fighter jets, even a full motion simulator has limits, another limitation is when too much emphasis is placed on the environment of the training scenario; we cannot simulate falling rain or dust in a computer simulation²⁰. These effects can be used, but it must be understood that their use is limited. PC Visual simulation based training seems like a good fit in terms of low-cost massive training for young people, especially if the goal of the training is basic cognitive skills.

One limitation in adopting PC-based Visual Simulations for training is acceptance by senior executives or officers who are used to live training and learning from real, not virtual, experiences. The major challenge is to prove that

its use will increase the process productivity, i.e., PC-based Visual Simulations reduce cost and increase capability²¹. A low cost development solution turns easier to proof this equation.

C. VISUAL SIMULATION DEVELOPMENT

1. Introduction

After looking at visual simulation uses, human factors and PC hardware limitations, we now focus on the visual simulation development process.

The process proposed is based on processes used in the game and animation industry. The model presented is a merge and simplification of two processes, one from an animation film production and the other from a video game development. These processes are closely related to the engine being used. The goal here is to present a process independent from a specific engine, to later identify the software tools and engine needed for each phase and to look at OSS/FS alternatives.

2. Development Process

The first process, presented in Figure 2, is from the Disney/Pixar's animation movie *Toy Story*, the first full-length, completely computer generated 3D animated movie²². The second process, presented in Figure 3, is from the *America's Army* game development¹⁷.

A major difference is that the animation production pipeline is aimed at offline video rendering performed by the camera department (Figure 2). In game development, the goal is a Game Level creation, or a scenario to be interactively played (Figure 3), and the video is rendered in real time.

In both there are common roles for animators, 3D modelers, sound engineers and artists. In the animation process programmers cannot be seen, probably because it is assumed that they are using a finished engine and are not adding any functionality to it. In game development, programmers hold a key position¹⁷. A point that is not presented in the game hierarchy is the design document creation, a starting point similar to the story and storyboard in animation.

Some common roles are:

- The project leader or director is in charge, managing the team and ensuring that all parts follow the guidelines expressed in the design document or story board.
- Programmers take care of the game engine development, maintenance and scripting programming (AI, new functionality), and put everything together to be rendered or tested.
- 3D Modelers create all the models, characters and objects.
- Graphical artists are in charge of the textures and lightning creations.
- Sound engineers create the sounds (if needed).
- Animators use motion capture tools to create animation scripts.

In small projects, roles can be merged to have, for example, a 3D modeler/artist and a programmer/ animator.

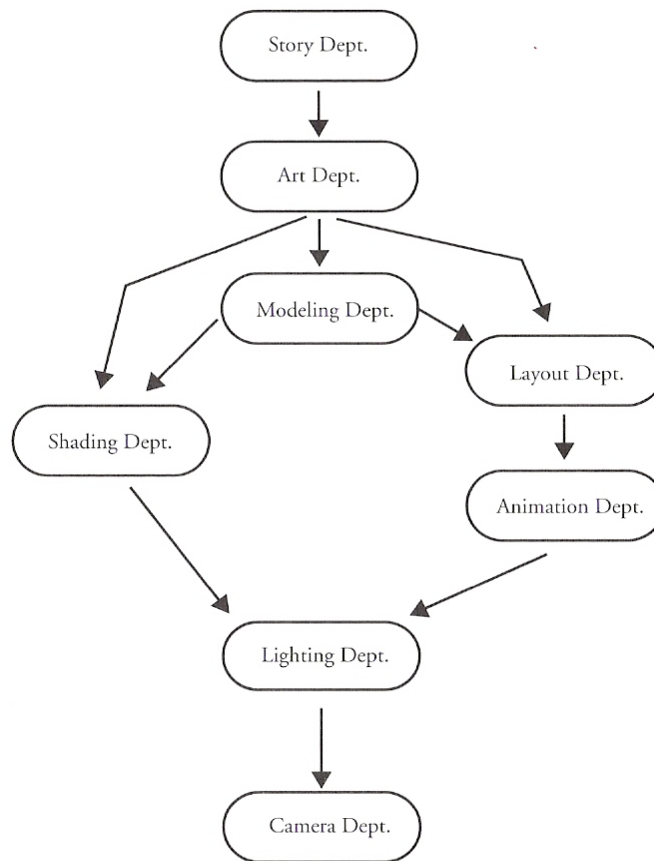


Figure 2. Animation Production Pipeline (From Ref. 22).

The game development pipeline from the *America's Army* project is presented below, it is presented as the hierarch relations between the team, but we can easily infer the process dynamic from the Figure.

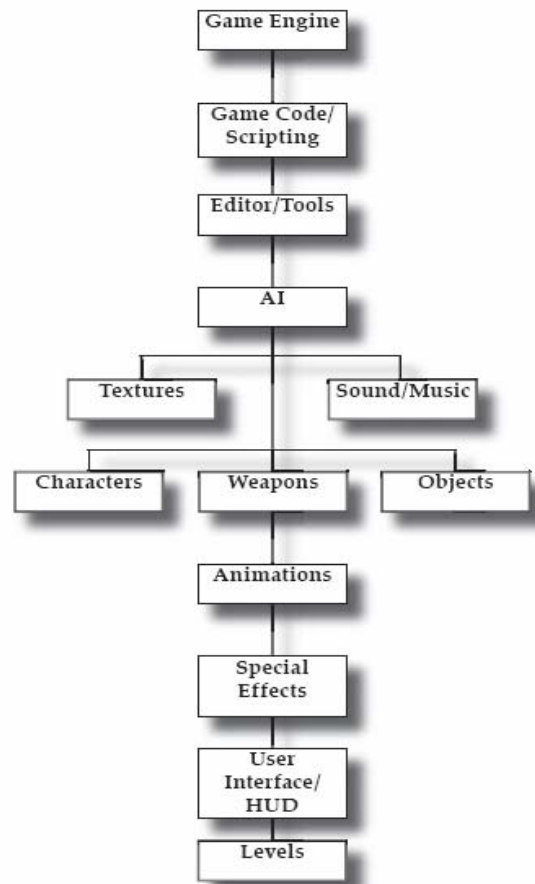


Figure 3. Game Development Hierarchy (From Ref. 17).

By simplifying and merging roles, a hybrid process is proposed to fit small developments (Figure 4). The roles of each position are generally explained and the goal is to create a process agile enough for a rapid development life cycle.

The OSS/FS tools to be used in that process are described in terms of features and licenses schemas in Chapter III.

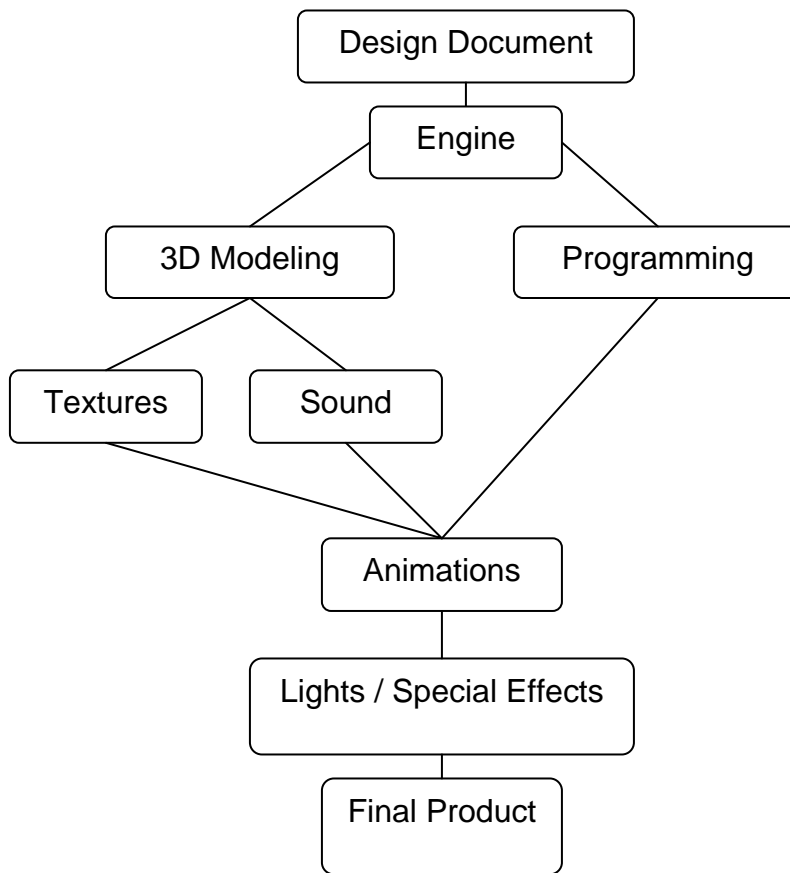


Figure 4. Simplified Development Process Proposed.

The project leader/manager supervises the entire process. The engine is chosen/developed depending on the requirements listed in the design document. The engine is the first tool needed and the one chosen will influence all other tools selected. There are proprietary and open source tools, and selection of the engine based on the open source environment is presented in Chapter III. Something that must be very clear at this point is that the engine is the base for the visual simulation or game development and after this choice is made, the project is tied to the selected engine almost forever.

Programmers work in engine programming, application development and animation. The development environment, or Integrated Development

Environment (IDE), is directly related to the OS and the engine, and will need compilers or interpreters (scripting) for the high level programming languages used.

Artists use a 3D modeler program, a graphical image editor, to create textures and special effects. Sound engineers use some kind of sound editing program to create the sounds for the simulation.

The final iteration in the development process is performed by programmers using the engine to put everything together, and compiling and creating the distribution package or rendering video if an off-line product is under development.

3. Authoring Tools and Engines

Types of tools and engines needed in every phase are listed below. Selections of specific programs are presented in the next chapter.

- Document Design and Management: regular office tools like spreadsheets, word processors and electronic mail programs.
- Programming: compiler, editor or IDE for the programming/scripting languages used in the engine with high-level functions and tools.
- 3D Modeling: 3D modeling program.
- Texturing: 3D modeling and image editor program.
- Sound: sound mixing program.
- Animation and Special Effects: High level functions and tools from the engine or in the 3D modeling program.

Normally a game\visual simulation development uses an engine, 3D modeling software, animation software, and an image editor. A sound mixer can also be included in the package to create or edit sounds, but sounds can also be externally recorded and introduced in the final product using the engine functions.

As already noted, the engine is the most important part and it is the layer that will connect all content created in the final game\simulation.

To illustrate the possible costs and options, some popular software tools are listed in the table below:

| Software | Utility | License Cost | Observation |
|-----------------------|---------------------------|-------------------------|---------------------------|
| UNREAL | Game Engine and Editor | \$7,999.00 + \$5,000.00 | Various licensing schemas |
| Visual Studio .NET | IDE | \$759.00 | Per machine |
| Borland C++ Builder X | | \$2,500.00 | Per machine |
| 3D Studio Max | 3D Modeling and Animation | \$3,145.50 | Per machine |
| Maya 6.5 | | \$6,999.00 | Per machine |
| Adobe PhotoShop CS | Image Editor | \$ 599.00 | Per machine |
| Various | Sound Mixing | \$ 25.00 | Average price |

Table 2. General Software Licensing Costs
(Prices verified on-line in July 2005.)

4. Development Problems

From Table 2, it can be inferred that visual simulation development is an expensive activity. Issues related to flexibility and personal costs must still be addressed. If the goal is massive training, low costs and versatile solutions are needed. The use of open source software is an attempt to achieve reasonable costs and flexibility, allowing the application of 3D visual simulations applications in new areas like military training.

It may seem that cost is the only reason to look at an OSS/FS solution but there is another important reason - the freedom to change and adapt the software according to training needs and independence from a unique vendor. These points are discussed in the next chapter.

In the *Handbook of Virtual Environment Technology*, Chapter 26, David Gross points out that a “killer” virtual environment application must appear to increase acceptance of the technology. The idea is to find an application as important to the commercial acceptance of VE as the spreadsheet was to the commercial acceptance of the PC. In fact, in areas like entertainment this application already exists in the form of games. The game industry is in some way pushing the hardware development when asking for better video cards and high speed CPUs. A more efficient and versatile visual simulation development that can be used in massive training could create a situation where a virtual environment “killer application” will be created in the form of a game-like interface for several types of training software.

One of the main reasons that limit the use of this technology is the high cost associated with building a game or simulation designed to learn. The expense associated with game engines is incurred before investments in developing content or learning return analysis, and engine dependency is created in the very beginning of the process. There are also cases where the engine became too expensive or the owner decided to stop its support. Transferring content and code from one engine to another is almost impossible³². The *Mythical Man Month* from Frederick Brooks stresses that expensive and large projects with many function points tend to fail³³, and visual simulations can turn into big projects. To use visual simulations for training, simple and inexpensive projects are needed. The selection of tools and engines is crucial from the very beginning in order to allow a project to be simple and flexible enough to be used as training or teaching tool³².

To make a project simple, flexible and inexpensive, based on the development process presented in Figure 4, the next chapter looks at OSS/FS

tools that can be used in each phase of the process. First, OSS/FS is generically presented and the most common licensing issues are noted, then a method for selecting and analyzing these tools is presented. Finally, tool and engine selections are presented.

THIS PAGE INTENTIONALLY LEFT BLANK

III. OSS/FS SOFTWARE FOR VISUAL SIMULATION DEVELOPMENT

A. OPEN SOURCE SOFTWARE

1. Introduction

This chapter is divided into three parts. In section A, a general analysis of open source software/free software (OSS/FS), including licensing problems and advantages, is presented. Next, in section B, a framework to select open source software projects is proposed. The last section lists OSS/FS tools to be used in visual simulation development, along with an open source visual simulation engine.

In discussing costs associated with game development, Michael Zyda, et al., in the paper “From Viz-Sim to VR to Games: How We Built a Hit Game-Based Simulation” (From Ref. 17), noted the reason to look at this kind of software when designing military training simulations:

...if we are really to follow the path towards game-based simulation, DoD needs an open-source game engine yesterday. DoD also needs to consider open sourcing the painstakingly developed art within its games, so teams don't throw scarce resources at reinventing 3D soldiers, weapons, and environments...

The initial motivation is economic, but the liberty to change, adapt and use other's work is also present in the text above. These are common points when analyzing open source initiatives - they do not directly cost money and there is liberty to change them if needed. For training/educational projects, these points look like a great advantage over proprietary software.

Here, a more radical idea is presented: a completely open source development, using not only free engines and 3D content, but also an open source set of tools during the visual simulation development.

Freedom is very important in the development process, but this freedom comes with a price. In OSS/FS sometimes there is lack of support, the user interface is less intuitive and there may be unreported bugs. For this reason, a

decision to adopt an OSS/FS tool in any project must be made with caution and the possibilities must be very clearly understood.

At this point it is necessary to understand the licensing schema involved because it can have a big impact on the final product distribution.

2. Open Source Licensing

One of the fundamental ideas behind open source licenses is to deny the exclusivity in the exploitation of a work. The purpose is to give the largest number of people available the opportunity and the freedom to distribute and change the code, thus creating a community of users or developers. This idea does have limitations, as any derivative work must have a kind of license consistent with the original software. Sometimes this point is enforced radically, like the GNU General Public License (GPL). Sometimes it is enforced in a more flexible manner, like the GNU Lesser\Library General Public License (LGPL). These two are the two major OSS\FS licenses, but there are several others, each one sponsored by different organizations²³.

According to Andrew M. St. Laurent, open source licenses make possible three improvements when compared to the traditional proprietary software licensing models²³. These improvements:

- Allow more possibilities for innovation - more people contributing in a project make the project better.
- Make the software more reliable because there are more people checking the code, finding and solving problems.
- Increase the longevity of a program because if a group or organization gave up controlling the development, another group or organization can catch up and continue the project.

The open source initiative (OSI) has parameters and definitions that allow the certification of a license as open source. These parameters are always updated, but basically there are 10 points that a license must comply with to be known as an open source license. These points are²⁴:

1. Free Redistribution: Paid or not, free as freedom.
2. Source Code Available: Distributed together or not.
3. Derived Works: Allow modifications.
4. Integrity of the Author's Source Code: Must be preserved.
5. No Discrimination Against Persons or Groups.
6. No Discrimination Against Fields of Endeavor.
7. Distribution of License: Allow derivatives distribution.
8. License Must Not Be Specific to a Product.
9. License Must Not Restrict Other Software.
10. License Must Be Technology-Neutral.

The most frequently used open source licenses are listed below. The goal of the list is to allow for comparison of their main characteristics, not to underline all the details of each licensing schema²³:

a. MIT / X License

This is a very simple license that originated at the Massachusetts Institute of Technology (MIT). Derivatives can be distributed in any way. This license can be used in commercial software or not and the source code could be available or not. There are practically no restriction to the use, redistribution or changing of the original code. It is also known as X license because it is used in the Linux X window system.

b. BSD License

A little more restrictive than MIT, this is an acronym for Berkeley Software Distribution. This license schema was created at the University of California, Berkeley. The licenses before 1999 had a problem related to the fact that every advertisement of software under that license was required to have a specific phrase giving credit to the original developer. As the software grew and became popular this list began to increase in a way that made commercial use difficult. This clause was removed to facilitate the advertisement and commercial

use of the software. Like the MIT license, derivatives work can be distributed in any way. The only restriction is to include credits from the original work. Derivatives can be released under the same license or not.

c. *Apache License*

This is a more elaborate license created by the Apache Software Foundation to release the Apache web server. The current version is 2.0 and was approved in 2004 in an effort to allow the use of the license in more projects outside the foundation and to create compatibility with the GNU/GPL licenses. Basically, this license is more explicit than the previous licenses, but still allows derivatives to use other forms of the license.

d. *GNU General Public License (GPL)*

The GPL, as with all the licenses created by the Free Software Foundation (FSF), has the limitation that all software derived from the original work must follow the same license, with the source code available for any modification by the users. This license is much more meticulous than the MIT, BSD and Apache. Its main concern is to keep the software free and available to everyone²³. This license limits the commercial use of the software and must be used with care. The common business model associated with the GPL license is to provide support or warranty for a certain program, or to distribute the software in a more convenient way.

e. *GNU Lesser General Public License (LGPL)*

The LGPL was created by the FSF for the specific case of software libraries developed to be linked to other programs. It is allowed to be used together non-GPL programs in certain ways. It permits the distribution of software under proprietary licenses linked to libraries under LGPL licenses - if the libraries were GPL the distribution would have to be GPL also. There are questions about dynamic and static linking regarding the use of the LGPL license that make its use sometimes tricky and complicated. Normally, the LGPL libraries must be distributed under their own license, with the source code available and static linked to other programs. In this case, the program that links to it can have other licenses.

f. *Mozilla Public License (MPL)*

This license was created by Netscape to be used in the Mozilla Project when the Netscape Communicator web browser was open sourced. It is a combination of ideas between GPL and BSD licenses²³. Programs copied or changed from MPL code must stay as MPL, but the same software can be combined with proprietary programs being released as proprietary code. The FSF imposes has restrictions on the use of MPL code together with GPL code. The Mozilla project is trying to release its code under a tri-license schema MPL/GPL/LGPL to solve that issue.

g. *Q Public License (QPL)*

This license was designed by Trolltech, a Norwegian company, to distribute the QT Toolkit. QT is a program used to develop Graphical User Interfaces (GUI), and it was the first license used to release the popular Linux GUI KDE. Because of concerns about the use of KDE as a non-GPL program, Trolltech changed the QT license to use both schemas, QPL and GPL. Derivatives can be distributed in the form of patches, so contributors can distribute their patches in different ways from QT itself. Today the company is encouraging people to use GPL instead of QPL and has created a QT Commercial License for proprietary software distributions²⁸.

h. *Artistic License (Perl)*

This license was originally created to be used with the Perl programming language (with some modifications). It is not used as much as other licenses because it is considered vague and confusing. Because of its vagueness, the FSF does not consider it a free software license. Programs developed using Perl cannot allow the modification of the standard Perl package itself.

A list of licenses is provided in Table 3, below. The Table details whether a license is compatible with GPL (i.e., whether code can be used together GPL code), and whether derivatives software must follow the same licenses. Examples of software released under the licenses are provided:

| | GPL Compatible²⁶ | Derivatives Work Licenses | Software under the License |
|-----------------------------|--|--------------------------------------|--|
| MIT/X | Y | Any | X Window System (X11) |
| BSD | Y | Any | OpenBSD, FreeBSD |
| GPL | Y | GPL | LINUX, MYSQL |
| LGPL | Y | Any with restrictions | Delta3D, OpenOffice |
| Apache | N | Any | Apache Web Server, Jakarta Tomcat |
| MPL | N | Any with restrictions | Mozilla Web Browser, Firefox Web Browser |
| QPL | N | QPL | Old versions of Qt Toolkit and KDE Desktop Environment |
| Artistic License | N | Any with restrictions | Standard Perl |

Table 3. Open Source Licenses

These licenses are listed to help the general understanding of the OSS/FS scenario and the rules applied to the programs that will be described in section C. There are more issues and concerns related to each license but further discussion of those issues is out of the scope of this work.

3. Problems of OSS/FS

There are some known problems associated with open source software and it is interesting to discuss them in the context of software development. The five points presented below were listed by George Weiss from the Gartner Group

when explaining the difficulties for Linux in penetrating in large enterprises²⁹. These points are not Linux exclusive and can be extrapolated to any OSS/FS project:

- OSS/FS is known for the potential for multiple source code distribution, causing fragmentation and interoperation issues. This statement is proved by the forkings and derivations of several OSS/FS projects. A development using an open source environment must control versions and upgrades in a safe manner, such as choosing more stable versions and delaying updates as much as possible.

- Higher support costs increase the total cost of ownership (TCO) with demanding workloads. In a teaching and training environment, the TCO can be significantly reduced by using OSS/FS, looking for standard packages and researching the better combinations of several tools.

- OSS/FS licenses could proliferate beyond the ability to manage them. During project development, the licenses of the tools and libraries that will be part of the final product must be well analyzed. The license schema chosen must conform to the business model intended for the project.

- Frequent software releases can create potential compatibility problems. This issue can be addressed by using the same initiatives presented before, version control and delaying updates (“playing safe”).

- Dependency issues, such as potential patent and copyright exposure, could raise risk management concerns. To address the dependency issue the software used must be chosen carefully with a method looking for mature and stable options. The copyright and patent issue is addressed by the license schema. One way to preserve secrets is keep the data closed but the application open.

- Usability is also an issue as open source software may require a more technical user approach. However, in an educational environment, where

one might expect to find more people with technical skills, this could be an advantage.

4. Open Source Advantages

The first item to address with regard to OSS/FS is cost. Using this kind of software does not mean the cost is zero, but it does create an opportunity to migrate the budget from buying software licenses to other development issues, for example, to increasing training.

Low cost is not the only advantage identified: in an education and training environment, flexibility and freedom from vendors are even more important. Open standards are another goal to pursue when looking to massive training.

5. Hardware and Open Source Driver Issues

The lack of hardware drivers, especially when using or developing visual simulations and graphics applications, cannot be forgotten. This used to be a major issue, especially related to Desktop Linux; today the scenario is a little better but, because the graphical cards market is driven by the game industry, and games are a major windows application, the latest video cards could not be automatically supported by the original vendor in an open source environment.

Even though Linux is being advertised as a supported OS by the two major graphics cards companies, ATI and NVIDIA³⁴, the community supported drivers are better options because they normally offer more stable and up-to-date versions.

The hardware configuration must be always under control, and upgrades and updates to the hardware and software must be carefully analyzed before being adopted.

During the test development presented in Appendix B, it is shown that after every update in the OS kernel (Linux Fedora Core 3), it was necessary to wait some weeks for the release of the new video drivers.

The approach recommended from that experience is to delay the updates and to maintain a current working version of the entire development environment

before going too far into a new version. This strategy can be applied for all the software and tools used during development.

In the next section, a framework to select and evaluate open source software is presented. Section C introduces the tools used to develop the Appendix B Visual Simulation Application.

B. SELECTING AND EVALUATING OPEN SOURCE SOFTWARE

1. Introduction

The evaluation and selection of open source software is not very different from a proprietary software selection process. The framework presented here is based on the proposal by David A. Wheeler³⁰ and follows four steps: Identify, Review, Compare and Analyze. This process was used in the selection of the tools and programs used in a visual simulation development and is detailed in the next section.

2. Identification of Candidates

The identification starts with the requirements list. Based on that list the candidates programs are selected. Again the 80/20 rule can be applied. It is difficult, if not impossible, to find even a proprietary program which satisfies all the requirements. If there is sufficient manpower to contribute to the project, customization possibilities can be looked as an option to add the functionalities not included in the program. This alternative can be very costly in the proprietary world.

Market share and popularity are good starting points to get a general idea about what is being offered. In OSS/FS solutions it is always important to look to the project activity and community support. The two the main repositories of Open Source projects are the websites SourceForge.net and FreshMeat.net³⁵, and both have mechanisms to verify the activity and popularity of a project. The inclusion of a program in the mainstream Linux distributions is another good indication that a mature program is under consideration.

3. Reviews

After identifying the candidates, the next phase is to look for reviews and tutorials about the programs previously selected. Market share or popularity can

be considered an indirect review and a good reference about a program³⁰. In the OSS/FS world it is sometimes difficult to measure the market share of a program because people can just download the product, use it and discard. Wheeler outlines an interesting way to analyze the market share of these programs by looking in a search engine like Google to see how many pages are linked to the project page. In Google this can be achieved typing keyword "link:" followed by the URL of the website in the query box (i.e., link: www.delta3d.org)³⁶. This can be used to gather information about people that not only downloaded the program, but are using it and referencing it from their own project pages. Another method for an informal review is to solicit input from professionals, colleagues and co-workers working in the field.

4. Comparison

This is the most difficult and time-consuming phase. The goal is to eliminate candidates and reduce the initial list for the analysis phase. Topics to be compared are frequently asked questions, documentation, examples, tutorials and mailing lists, etc.

The dynamism of OSS/FS and the forking phenomenon (when a project splits in two different projects) can make comparison difficult. The following list of attributes extracted from Reference 30 can be used in the comparison process:

- **Functionality:** The integration with existent software, hardware, OS and other programs used during the development must be analyzed. Few programs have all the functionality needed, and normally they are used together with other programs. In OSS/FS, it is possible to add functionality, this is a differential, but can increase development cost and time. The effects of such additions must be weighed before a final selection is made. OSS/FS provide also the possibility to add functionality; this is a differential, but doing this increase development cost and time and must be weight before the final selection.
- **Costs:** OSS/FS does not mean the program is necessarily free, with no cost: it means that there is freedom to change and use the program. What must be compared is the Total Cost of Ownership (TCO). TCO includes all the costs

related to using, distributing and maintaining a program, including factors such as manpower, staff qualification and training.

- **Market Share:** Verification of how popular the program is. It may be difficult to verify popularity in the OS/FSS world, but it is an important indication of a program's maturity and stability.

- **Support:** OSS/FS software can have paid support provided by different vendors, this is one of the business models used. The starting point to compare support could be the project website and the list of organizations contributing to the project. They are the first option for software support. The second option is large community projects, where the support is the community itself. In the case of community support, there are some risks involved because of the informal and sometimes chaotic way the communities are formed. One possible way to reduce the risk is to rely on community support only when there is enough knowledge in-house about the software³⁰.

- **Maintenance:** Maintenance is directly related to the longevity of the project and it has the same options as support. A major advantage in the case of OSS/FS is that the self-maintenance is facilitated by the open nature of the source code. If the project becomes stalled, it is possible to turn over maintenance to an in-house team. Of course, as mentioned before, there are costs and time implications involved.

- **Reliability:** To test software reliability, the option is to create pilot cases and real work load tests. Mature projects tend to be more reliable than fresh ones. SourceForge.net and FreshMeat.net³⁵ both have mechanisms to measure the maturity, and by inference, the reliability of projects. Another comparison point is the inclusion in the most popular Linux distributions.

- **Performance:** The only way to compare performance is to undertake runs with real world data. In the specific case of visual simulations, this requires loading different scenarios and 3D databases, and then testing framerates in different hardware configurations.

- Scalability: Can be compared by studying real case scenarios. It is basically a case study comparison. If there is no known case to be studied the pilot test is an option.

- Usability: The learning curve of a program is directly proportional to its usability. Today's applications must have a GUI because GUIs are the most acceptable way to interact with a program. If the analysis is about an API, a GUI is not relevant, but the way it interacts with other programs, libraries and OSs is a key to assessing its usability. There are several GUI guidelines. In Linux, the two most commonly used GUIs are the KDE and GNOME, and both have guidelines listed on their websites³⁷. It is interesting to compare which guideline is followed by the program of interest. A usability test is an option if more information is necessary.

- Security: One could assume that the possibility to search and read all the source code in OSS/FS will allow the identification of any security issue, but this is not viable option because the size of the code involved can easily reach thousands of lines. A good hint about security and reliability can be obtained by looking at who and where the program is developed. Looking at users may also be necessary, and if still not enough, a deeper security analysis must be performed.

- Flexibility/Customizability: This is the great OSS/FS advantage over proprietary software; OSS/FS can be modified as necessary, in house if there is sufficient knowledge or through several external contractors. If customization is necessary, the associated costs must be considered.

- Interoperability: To compare interoperability it is necessary to look at what standards are followed by the program and if they are in accordance with the standards used in the project development, such as file types, OS, network protocols and so on.

- License/Legal Issues: The license schema must be carefully analyzed, understanding the type of licenses described in section 2 of this chapter. The licenses used, especially in libraries, to be incorporated in the project must be

in accordance with project guidelines and future intentions. External help may be necessary to compare all licenses and contracts issues. This is more a developer than an end user concern.

- Other: Public policies and guidelines are relevant every time an application or program is used, especially in a military scenario. The MITRE Corporation prepared a study for the Defense Information System Agency that recommends the use of OSS/FS in the Department of Defense (DOD) for application development. Several tools and libraries, as well as licensing issues, are addressed³⁸ in this study.

5. Analysis

The evaluation phase is the time to deeply evaluate the topics previously presented and to perform a last analysis before making a final decision on the software used.

For example, one point that can be better explored is to identify who controls the project development. There are reasons that development controlled by a formal organization is preferable. Community projects depend on unpaid staff and group consensus; they have a slow evolution and tend to have fewer people working in some areas, such as documentation and support. A formal organization can hire more people as needed and there is no need to build consensus to plan a project roadmap. This is an advantage presented in the Delta 3D Visual Simulation engine that is described in the next section. On the other hand, when there is dependency on a single organization to manage a project, their roadmap has to be followed and a dependency can be created, similar to the dependency involved when using proprietary programs. Community projects can be constrained by lack of volunteers or management direction; formal organizations can be constrained by budgets and profits.

The visual simulation technology and development process was presented in Chapter II. OSS/FS selection and evaluation was presented in previous sections of this chapter. The tools, libraries and API used to develop a visual simulation are presented in the final sections of this chapter. The visual simulation developed is

described in Appendix B, which also illustrates the concepts and provides examples of the OSS/FS tools used in 3D graphics development in a complete OSS/FS environment.

The figure summarizes the selection and evaluation process:

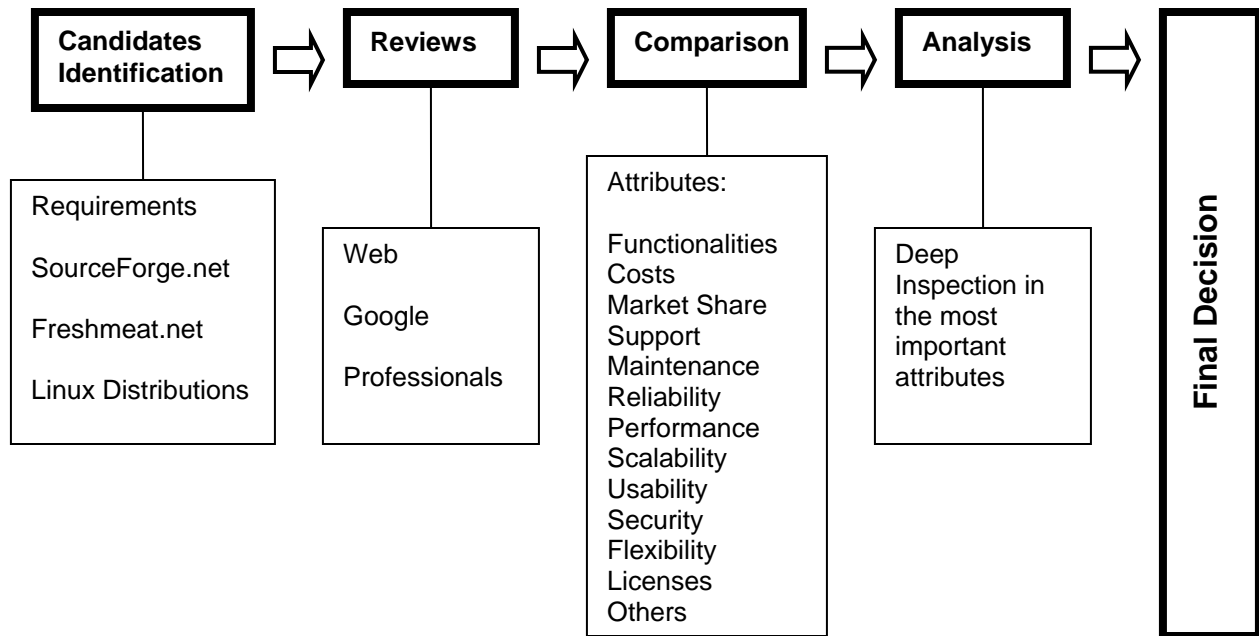


Figure 5. Evaluation and Selection Process.

C. OPEN SOURCE TOOLS AND ENGINES FOR VISUAL SIMULATION DEVELOPMENT

1. Introduction

The hardware boundaries in the scope of this work were set in Chapter II. The hardware used is a COTS personal computer with regular input devices (keyboard and mouse) and a regular display in a reasonable size (17"). In the PC world, when looking to open source technology, attention automatically turns to Linux. This is the open source OS of choice because of its stability and support. There are several distributions available, some aiming at servers, others at desktop users and so on. We compare two distributions that rely on different business models: Fedora Core and Debian.

Fedora Core 3 has community support and a formal organization backup provided by RedHat, one of the primary Linux vendors. Debian, on the other hand, is a completely community supported distribution. In the development presented in Appendix B, the selection of Fedora Core 3 was driven by the support available (community plus a company), stability, tools available, and existence of pre-compiled versions of the libraries used. The Fedora Core 4 release was not selected in order to stay one step behind the latest version and thus be able to evaluate stability. Appendix A includes tutorials and references to install this distribution and set a development environment.

The architecture chosen is a regular PC running Linux OS (Fedora Core 3). Tools, engines and libraries are selected based on this first platform option and following the process presented in Chapter II, section C.

Some of the tools are already part of the Linux distribution package, such as the GCC compiler, Python scripting language, Emacs text editor and GIMP image editor. Other tools were selected later. The hardware drivers need to perform the 3D graphics (OpenGL) and the audio drivers (OpenAL) have to be installed and updated.

The drivers are points that can turn the OSS/FS development option harder than a proprietary solution. They must be very well analyzed based on the particularities of the hardware and OS used. Having installed the drivers and selected the first tools and programming languages, the next phase is to choose the simulation/game engine, which may be the most important decision in the project. After this point, all development will be tied to the engine selected.

Delta3D⁴² was selected based on support availability and compatibility with the OS, languages and environment. The strong institutions supporting Delta3D compensate for the lack of community support. The well-known libraries in its dependencies list are also a good indication of stability, even in a situation where the engine cannot yet be classified as a mature project. Its biggest problem may be the lack of case studies and projects using the engine, but other OSS/FS engines are in the same level of maturity. For example, Ogre3D⁴⁸ cannot be

considered a full engine; it is more a 3D scene render engine and it lacks some functionality needed, such as physics and sound, being supported only by the community. Panda3D⁴⁹ is supported by formal organizations but is aimed more at entertainment games than visual simulations. Panda3D has one successful case: the game that originated the engine.

Table 4 lists each phase of the process and the general OSS/FS options made with the related proprietary tools:

| | OSS/FS | Proprietary |
|-------------------------------------|------------------------------------|--|
| Management and Documentation | OpenOffice | Microsoft Office, Corel WordPerfect Office |
| Engine | Delta3D | Unreal |
| Programming Tools | GCC Compiler and EMACS text editor | Microsoft Visual Studio .NET |
| 3D Modeling | Blender, Wings | Maya , 3D Studio Max |
| Image Editing | GLIMP | Adobe Photoshop |
| Audio Editing | Audacity | Adobe Audition |
| Special Effects | Delta3D Engine Utilities | Maya , 3D Studio Max |

Table 4. Proprietary and Open Source Tools

Now only the tools directly related to the visual simulation development will be detailed. Generic tools like the compiler, text editors and office tools are considered out of the scope of this thesis, as they are not specifically used for visual simulations developments, but for any software development.

2. OpenGL

OpenGL₃₉ is the acronym for Open Graphics Library. It is a low level graphics API that originated from the IRIS GL created by Silicon Graphics (SGI)⁴⁰. Today, OpenGL is the most popular low-level graphics API and the only one that

allows Linux/Windows portability. It is being used in the development of interactive 2D and 3D graphics applications. Because of its popularity, OpenGL is very well supported by the graphics cards vendors and has hardware drivers for most of the Linux distributions.

The first OpenGL standards were founded with the creation of the OpenGL Architecture Review Board in 1992, which concentrated the major players in the computer graphics world. It allows rendering, texture mappings, special effects and other visualization functions. There is plenty of on-line documentation and books about OpenGL programming being a natural choice for Open Source developments together with Linux.

3. OpenAL

OpenAL⁴¹ is the audio counterpart to OpenGL. OpenAL stands for Open Audio Library. It is a 3D audio API used with gaming and simulation applications. Like OpenGL, it is cross platform and well supported under LINUX. Its architecture is a little more modular than OpenGL and the main reason for its creation was the need to standardize sounds API.

It is supported by many hardware vendors and has strong support from the main audio hardware player, Creative Labs. Though not as popular as OpenGL, it still has good community support and hardware drivers for several OSs. It is being used in several multiplatform commercial games

4. Delta3D Visual Simulation Engine

The selection of the Delta3D⁴², as already stated, is based primarily on the functionality and support available, but also on its development dynamism and license characteristics. The version used in this research was the 0.8.6

Among the organizations supporting Delta3D are the Moves Institute, the Naval Education and Training Command, BMH Associates and the Marine Corps Program Manager for Training Systems. Its website now has 304 registered users, the average number of unique visitors per month is around 5,000 (since July 2004) and the program has been downloaded from SourceForge.net by 13,321 users as of July 2005. These numbers indicate that a community is starting to form.

The engine itself is a thin layer created on top of several open source libraries, API, and projects. It provides a set of classes that make difficult tasks from the low level libraries easier to program using the higher level functions and classes. A differential in the project are the utilities provided together with the engine to help the development process. These include a 3D model viewer/converter, Graphical Particle Effects Editor, BSP Compiler, HLA Graphical Stealth Viewer and, in future versions, a Scene Level Editor.

The diagram in Figure 6 shows the low level libraries the Delta3D version 0.8.6 depends on.

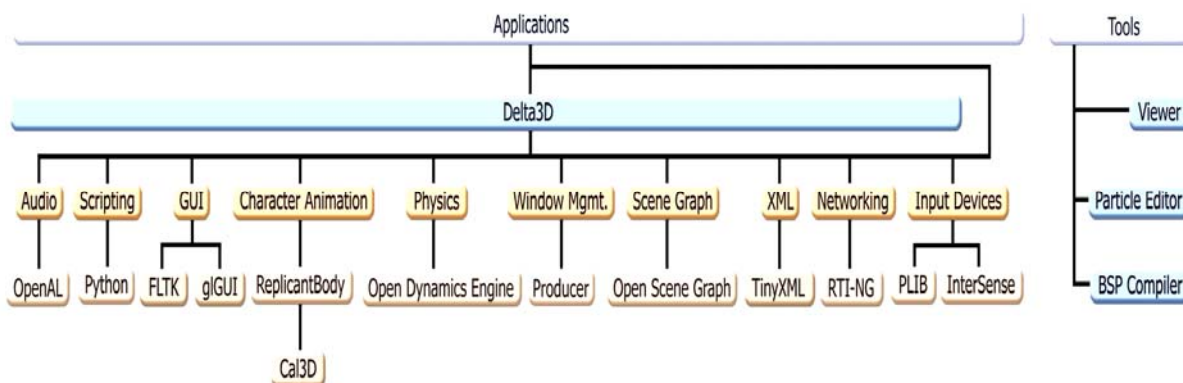


Figure 6. Delta 3D Dependencies Architecture.

This project is a good example of the use of OSS/FS in visual simulations – it is like wrapping a library around several other open source libraries. The problem with this strategy is that while it brings the best of the open source world in terms of functionality, it also brings all the problems and deficiencies from the low level libraries, which can compromise some of the functions needed. This could be diminished as the project becomes more mature and interaction with the community increases. A strong interaction among the project development team and the dependencies libraries communities could greatly improve the stability of the project. The licensing schema is LGPL, which provides enough flexibility to be used in both the OSS/FS and the proprietary world.

Some of the dependencies libraries and tools of the project that were used in the test development are detailed to understand the kind of libraries the engine relies on and their utilities.

a. *OpenSceneGraph*

OpenSceneGraph⁴³ is a rendering system that establishes a scene graph to render a 3D scenario. It was created by Don Burns when he was an engineer at SGI to support a hang-glider simulator. Robert Osfield was also an early contributor. Today, both provide support through their own companies. OpenSceneGraph is very popular and is being used by several developers to create visual simulations, games, scientific visualizations and modeling. There are several successful cases that denote a good maturity level. It is written entirely in Standard C++ and OpenGL and runs on several platforms and operating systems. Choosing OpenSceneGraph is a natural decision when looking to the available open source rendering systems because of its support, maturity and features. It is licensed under LGPL, which is a very convenient schema for use in the engine itself. It can be considered the most important dependency of the engine providing the basis to Delta3D.

b. *Open Dynamics Engine (ODE)*

ODE stands for Open Dynamic Engine⁴⁴. It is an open source real time physics engine formed by two components, one for rigid body dynamics simulations and the other for collision detection. It is a high performance library allowing good computer simulation of rigid body dynamics. It provides a C++ API that is wrapped by the Delta3D high-level classes. ODE is useful for simulating vehicles and objects in virtual reality environments and games. It is currently being used in many computer games, 3D authoring tools and simulation tools, indicating a reasonable maturity level. It is platform independent and licensed under LGPL, like OpenSceneGraph.

c. *Delta3D Viewer*

A model viewer is a common feature in visualization engines and is used to allow the inspection of static models before including them in a scene. This viewer is a very handy tool in the previous analysis of 3D models allowing to check

the file formats supported by Delta3D and OpensceneGraph like: .3dc, .3ds, .ac, .dw, .flt, .geo, .ive, .logo, .lwo, .lws, .md2, .obj, and .osg. The tool also permits the inspection of textures, lights and orientation in the model and the conversion to osg or ive.

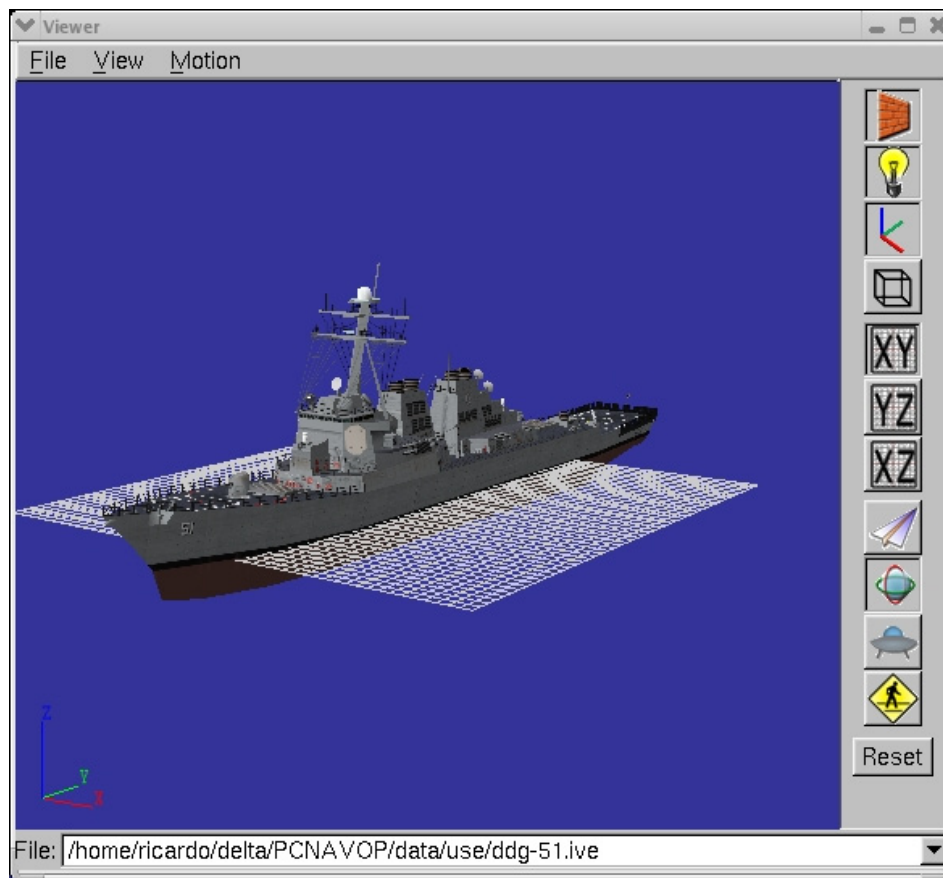


Figure 7. Delta 3D Viewer.

d. Delta3D Particle Editor

This editor is a real time graphical particle systems editor used to create OSG particle systems. This program facilitates the development and creation of particle systems, allowing real time visualization of effects. This tool, introducing functionalities not easily found in other tools, can be considered one of the best tools delivered with the engine.

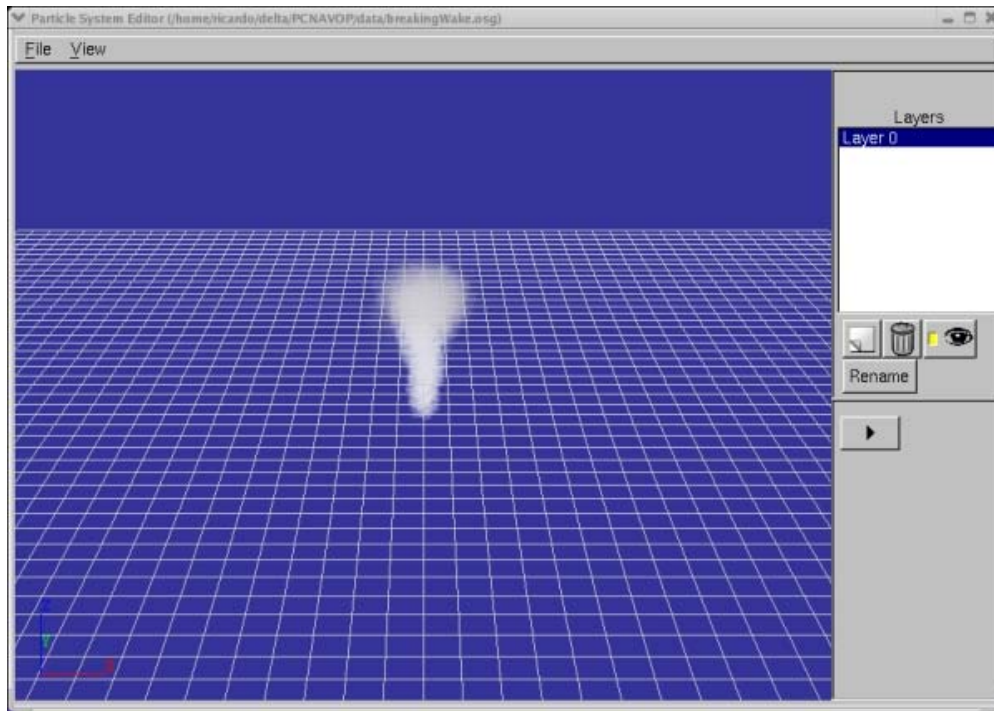


Figure 8. Delta 3D Particle Editor.

e. STAGE (Simulation, Training and Game Editor)

STAGE is the Delta3D Graphical Level Editor. It is a promising tool that may improve and increase the use of this engine. It is used in conjunction with the other tools and has the potential to speed up the development cycle. The editor allows a visual construction of a scene world, including terrain, objects and actors. It is intended to be an OSS/FS version of the UNREAL Editor based on the Delta3D engine. It has the same types of viewports used by proprietary engines to view, manipulate and navigate the world created. The program was not used in this work because it was released after the selection of the version 0.8.6 and there was not enough time to test the tool.

f. Other Libraries and Dependencies

As shown in Table 5 there are many other libraries used as part of the engine. They are listed below with the versions used in the last engine release, but they will not be detailed here as they were not directly used in the simulation developed. These libraries were used only through high-level engine classes.

| Library | Version | Function |
|-----------------|-----------------|------------------------|
| CAL3D | CVS: 2005-03-14 | Animation |
| Crazy Eddie GUI | 0.3.0 | GUI System |
| glGUI | CVS: 2004-03-09 | GUI System * |
| FLTK | 1.1.6 | Windowing System |
| GDAL | 1.2.6 | Geospatial data reader |
| InterSense SDK | 3.5.8 | Input devices driver |
| TinyXML | 2005-07-25 | XML parser * |
| Xerces – C++ | 2.3.3 | XML parser |
| Replicant Body | 1.8.4 | Animation |

Table 5. Delta3D Dependencies Libraries List

* These libraries are scheduled for deprecation in future versions.

5. GIMP

GIMP⁴⁵ is an acronym for GNU Image Manipulation Program. This program was developed by two students at the University of California Berkley; version 2.0 is now in release and the program is distributed under GPL. The program was previously criticized for its difficult installation (it was only source code distributed) and bad user interface. But after version 2.0 the reviews changed, indicating that the program is a major player in the image editing scenario. It allows all common image manipulation tasks such as photo retouching, image composition and image authoring, and it is included in almost all LINUX distributions, indicating a very good maturity level. It is also very portable, having several OS and languages versions.

For basic and some advanced texturing, GIMP is a tool that can completely substitute for proprietary programs like Adobe Photoshop and Jasc PaintShop

without loss of functionality. It is also well supported with literature and a lot of web references. GIMP can be considered the basic image editor for the Linux and OSS/FS world.

6. Blender

Blender⁴⁶ is an open source 3D modeling and animation tool that also has a built-in game engine. The program was used and analyzed only in the scope of its 3D modeling capabilities. The tool was used to edit and create 3D models that were exported using one of the external plug-ins. This external plug-in feature can be extended, adding new functionalities. The plug-ins are developed using the Python scripting language and there are several 3D file formats importing/exporting plug-ins already available. In the particular development described in Appendix B, the OSG and 3DS exporters were successfully tested.

The tool development is led by the Blender Foundation, a virtual foundation currently without official offices, which is chaired by Ton Roosendaal, a Dutch programmer who originally created Blender for a private company and later bought the code and open sourced it. Blender is distributed via a publicly accessible source code system under GPL, and it is well supported, having several references in literature and web sites, including tutorials and examples.

Even without all the functionality and support of the major 3D authoring tools for games and visual simulations, such as 3D Studio Max (Discreet) and Maya (Alias/Wavefront's), the features provided are sufficient for a small development and powerful enough to be used for training simulations. The lack of pre-built models can be overcome using the plug-ins. The learning curve is not much different from similar programs and, in some cases, could be considered even faster.

7. Audacity

Audacity⁴⁷ is the program used to create or record the simulation sounds (audio editor). There are plenty of audio editors available in both the OSS/FS and proprietary world: the selection here is based on reviews and functionality. Audacity is a very well reviewed open source program distributed under GPL. It is

available for Linux and other operating systems. It supports a variety of file formats and has more than enough features for a medium sized project.

Even lacking some pre-recorded audio effects, the effects available are good enough. The very simple user interface, not so common in open source projects, is a plus in this tool. Like GIMP, there is no loss of quality or functionality when using Audacity compared to proprietary options like Adobe Audition.

IV. ANALYSIS, CONCLUSIONS AND FUTURE WORK

A. ANALYSIS

The architecture proposed to develop a visual simulation using OSS/FS is described in Figure 9 below:

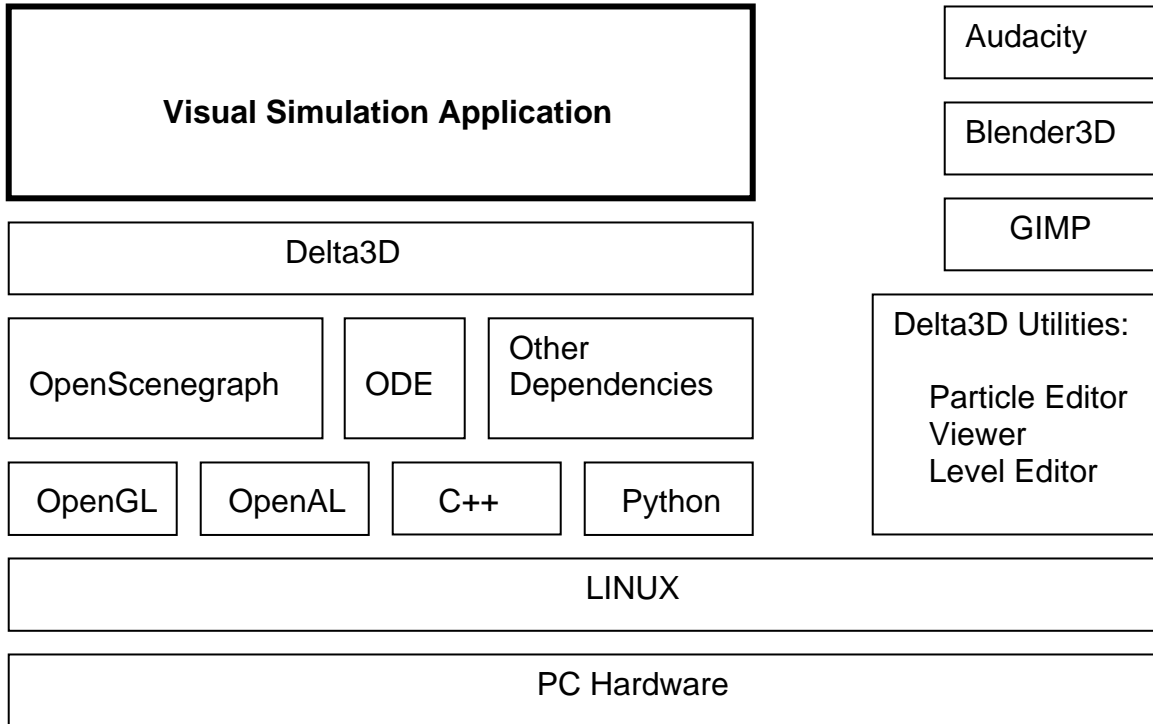


Figure 9. Architecture Proposed.

To develop a visual simulation to be used as a training tool, the basic hardware presented is a COTS PC, with at least a 17" video display and stereo sound. This configuration, even having limitations, is considered the minimum needed to stimulate the auditory and visual senses of a trainee and achieve a reasonable sense of immersion and training transfer.

The operational system is a well supported Linux distribution with hardware drivers for OpenGL and the software drivers for OpenAL installed. The basic

programming languages are C++ and Python, normally available in any Linux distribution, including development tools (compilers and editors).

OpenSceneGraph and ODE, and their dependencies, can be considered the two main libraries used by the engine. On top of these libraries and other dependencies libraries goes the engine. The application was developed using the high level functions and classes of the engine and, only if necessary, accessing the low level libraries to add advanced functionalities or enhance the visual presentation.

The engine utilities (viewer, particle editor, etc.) and the other programs, Blender, Audacity and GIMP, were used to edit and create content. These programs have Linux, Windows and other OS versions, and if needed, they can be used in other environments. In fact, content creation is a task that can be externally developed and incorporated to the application later; all that is needed is to be aware of the file formats supported.

The main problem with using a complete open source environment is controlling version/updates in all tools used, including all the dependencies, starting at the very bottom of the stack, the OS kernel. A good strategy is to freeze updates when a smooth working architecture is reached. Before upgrading, it is important to carefully analyze the improvements of a new version. This rule applies for programs as well as libraries.

The time lost configuring an OSS/FS environment is much higher than in a proprietary configuration and can be considered the “liberty” price paid, so every upgrade has the potential to be much more time consuming than desired.

B. CONCLUSIONS

A visual development process using this OSS/FS architecture has the potential to be less costly than a traditional windows development. The time frame of the several development iterations could be higher in the beginning of the process, but tends to reduce as the project and the tools used evolve. What connects all the phases is the engine, which, being open source will always permit

enough freedom to execute changes and adaptations as needed. Again, the time price must be seriously considered.

In some visual simulation application cases, like low COTS massive training, this model seems to fit very well. It is clear that a PC cannot be used as a substitute for all kinds of VE applications, but its use must be seriously considered, especially when costs constrain the project. The limitations are introduced basically by the hardware and not by the software or tools used.

Applications, like the one described in Appendix B, can be developed and open sourced in such a way that well trained small development teams will also be able to change, customize and redistribute as needed. Proprietary or classified data can be put outside the application itself, in the form of pre-built 3D models or XML data files.

One way to overcome the update and version problems is to distribute the training application itself as a bootable CD containing the OS and precompiled versions of all the drivers, dependencies and the engine. This approach can also be applied to test training solutions before jumping into more expensive alternatives.

C. FUTURE WORK

This study provides the basis for future visual simulations development using open source tools. It supports the spread of some OSS/FS programs, like the Delta 3D visual simulation engine, to develop low cost educational and military training applications. Possible future research includes using the proposed tools and framework in several developments to better evaluate these options, and testing OSS/FS developed game-like applications in training. The analysis of the scalability of the architecture is also a interesting point for future works.

To address the version compatibility issue and facilitate application distribution in an almost windows exclusive world, the creation of a basic bootable CD distribution (known as Linux LiveCD), including the libraries and programs needed, is another possibility for future work that might be generated from this thesis.

A strategy in that direction is the creation of some type of distributions such as:

1. **Development Version:** Including OS kernel, GUI, video drivers, content creation tools, compilers, editors, engine and CD-masters to create the application distribution bootable version. This option may fit better in a hard drive installed version than a bootable one, but with enough read-write media in the form of a network share or USB device, the bootable version can also be used.

2. **Application Distribution Version:** This is a simplified version of the previous one and does not even need a GUI. Only the kernel, pre-compiled versions of the programs and network support as needed, so the training could be performed with minimal interference to the other activities for which the trainee machines are used.

These solutions could create a flexible and interesting way to distribute training applications to run on a variety of PC hardware. The budget will be concentrated in the application development and distribution processes and not in licenses or hardware acquisition.

APPENDIX A. INSTALLING AND SETTING LINUX AND DELTA 3D

This tutorial describes the installation and settings for the LINUX distribution FEDORA CORE 3 and DELTA 3D Gaming and Simulation Engine version 8.6 in order to setup a visual simulation workstation. If a different LINUX distribution is used, the settings could be similar, but more references are needed.

INSTALLING AND UPDATING LINUX FEDORA CORE 3

Before installing Linux, it is a good practice to look for specific references on the web about the distribution and hardware being used.

Instructions to download FEDORA CORE 3 and prepare the installation disks are available at <http://fedora.redhat.com/download/>. There are several possibilities for disk partitions and physical installations that are not detailed here because, at this time, the choice about how the system will work has been made: dual boot, single system, one hard drive, two hard drives etc. Having made the decisions, in Fedora, the Anaconda installation software takes care of the process. The choice made during the installation process was “**workstation installation**” (Figure 10) one time the system was to be used for software development. This option automatically installs the compilers for C++ (GCC) and Python and other editing tools.

The default desktop in Fedora is GNOME, but an interesting option is to use KDE and its development tools, also installed. In this tutorial the screen shots are all from KDE.



Figure 10. Fedora Core 3 Installation Type.

After the installation, it is necessary to update your system and download more software. To do this automatically, a good option is to setup the yum program (instructions are available at <http://www.fedorafaq.org/#yumconf>). According to the references at <http://www.fedorafaq.org>, the default servers from RedHat are slow and changing to the yum settings is recommended. When everything is ready to update the system, the graphical tool from the desktop (KDE or GNOME), or the command line below can be used:

```
yum update
```

(Note: Root privileges are needed to install and update system files and programs. Linux good practice is to never login as root and only use root privileges when needed. This can be done by creating a regular user and typing the command **su -** to assume root identity when needed.)

TESTING AND INSTALLING OPENGL DRIVERS

After the installation, the video card hardware accelerated OpenGL drivers must be checked by typing the following command:

```
glxinfo | grep direct
```

If the answer is “**direct rendering: yes**,” the OpenGL hardware accelerated drivers are already installed. If not, it is necessary to install them manually.

Pre-compiled versions of the drivers for several Fedora kernels are available at <http://rpm.livna.org>. To identify which kernel is being used, enter the command below (the answer is the kernel number):

```
uname -r
```

References about the installation of the OpenGL drivers for the most popular video cards are available at <http://www.fedorafaq.org>.

ATI: <http://www.fedorafaq.org/#radeon>

NVIDIA: <http://www.fedorafaq.org/#nvidia>

The graphics cards vendors do not normally provide kernel specific RPM versions and it is necessary to download and compile the source code for your distribution. At <http://rpm.livna.org>, again, specific drivers for several Fedora kernels are available. To automatically look for the kernel version, download and install the drivers, the command below can be used:

```
ATI:      yum install ati-fglrx kernel-module-fglrx-`(uname -r)`  
NVIDIA:   yum install nvidia-glx kernel-module-nvidia-`(uname -r)`
```

The drivers will become active after the next boot.

After every kernel update it is also necessary to update the drivers. To update the drivers, the following commands can be used (this must be done BEFORE restarting the system with the new kernel):

ATI:

```
yum install kernel-module-fglrx-`(rpm -q --queryformat="%{version}-%{release}\n" kernel | tail -n 1)`
```

NVIDIA:

```
yum install kernel-module-nvidia-`(rpm -q --queryformat="%{version}-%{release}\n" kernel | tail -n 1)`
```

Once the OpenGL drivers are “up and running” the next step is to install Delta3D.

INSTALLING DELTA3D

From now on, the screen shots and mouse commands are from KDE’s Konqueror file browser. Command lines will also be presented.

First, the Delta3D Linux distribution package must be downloaded (<http://www.delta3d.org/index.php?topic=downloads>) into a “youruser” home folder (/home/youruser):

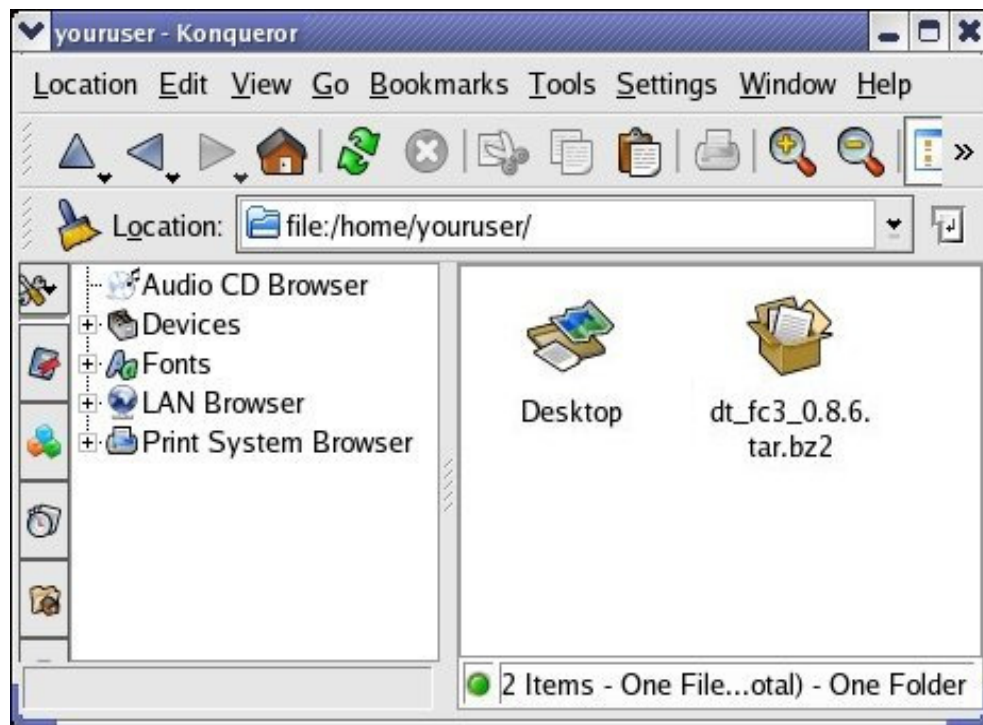


Figure 11.

Delta3D Package Downloaded.

Then the package must be extracted by right clicking the mouse and selecting Extract, Extract Here:

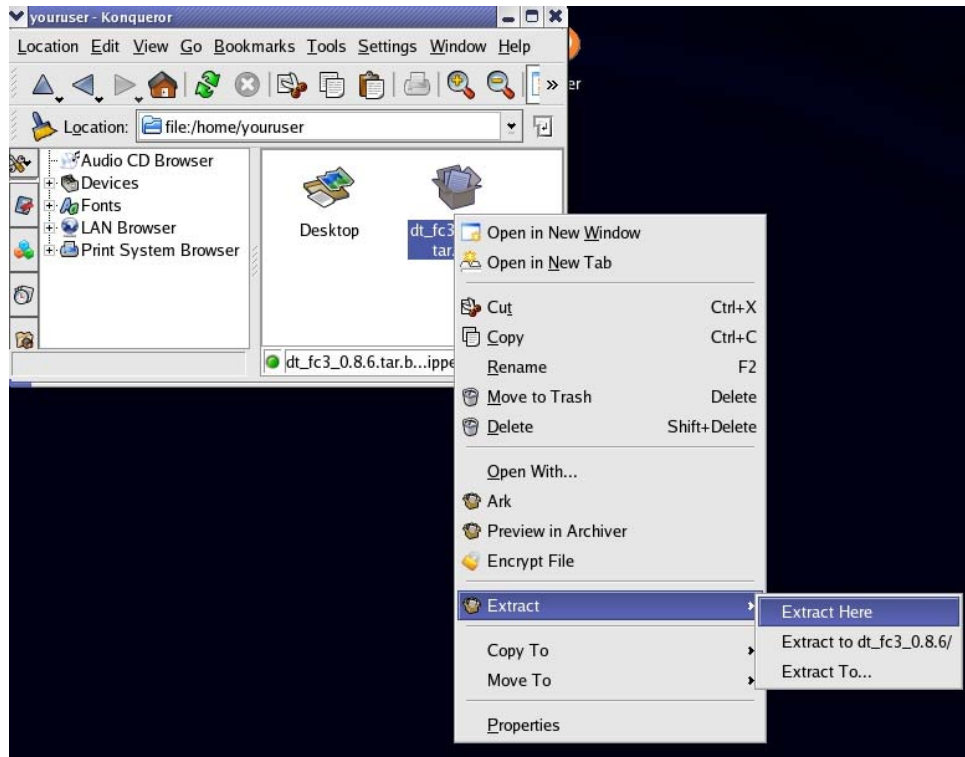


Figure 12. Extract Delta3D Package.

Command line:

```
gunzip dt_fc3_0.8.6.tar.bz2
```

Under the folder /home/youruser a new folder named delta3d will be created. This folder will contain the file structure presented in Figure 13.

To get there:

```
cd delta3d  
ls
```

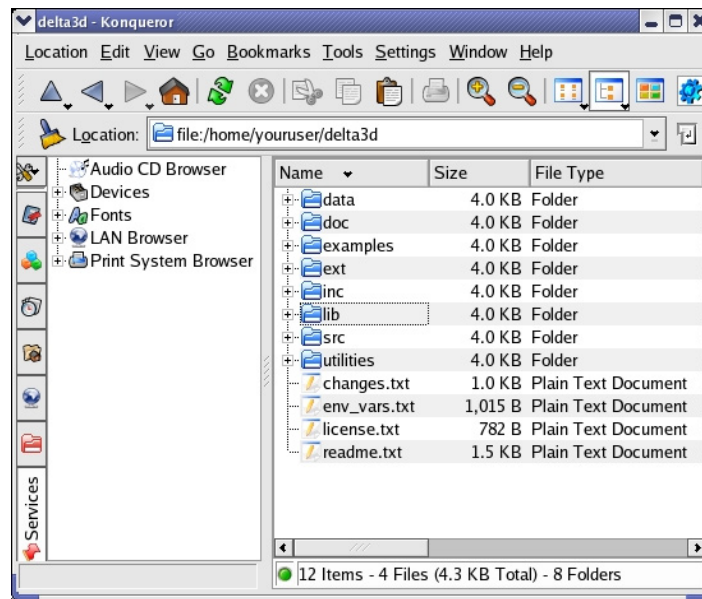



Figure 13. Delta3D File Tree Structure.

(Note: This view can be selected using the tree view button on the Konqueror Main Toolbar )

This file structure corresponds to the following:

lib: Delta GCC 3.4.2 compiled libraries.
 inc: folder with Delta3D Include header files.
 data: Models and Files used in the examples.
 doc: folder to generate the documentation using doxygen.
 examples: Several example programs.
 ext: External dependencies: (versions.txt show the versions used)
 inc: external dependencies include header files.
 lib: GCC 3.4.2 compiled external libraries.
 src: Source code for Delta3D.
 utilities: Some programs to help the development:
 Graphical Particle Effect Editor
 3D Model Viewer
 BSP Compiler
 HLA Graphical Stealth Viewer , and
 env_var_setup.exe to windows environment variables installer

The file named **SConstruct** in the main folder, and several **SConscript** files in folders below, are used to compile and link Delta3D libraries and examples using the software construction tool SCONS (<http://www.scons.org>). These files are Python scripts used as "configuration files" to build the software. The installation of SCONS is described later. The other files are READMEs, Projects and Solutions to be used with Windows Microsoft Visual Studio and settings for the windows environment variables.

Before compiling Delta3D and running the examples, the environment variables must be set. In Linux this could be done in several ways, one of which is to edit the **.bashrc** file. This file is a hidden file inside the **/home/youruser** directory. Below is the command to view:

```
ls -a
```

or select **View, Show Hidden Files** in the file browser.

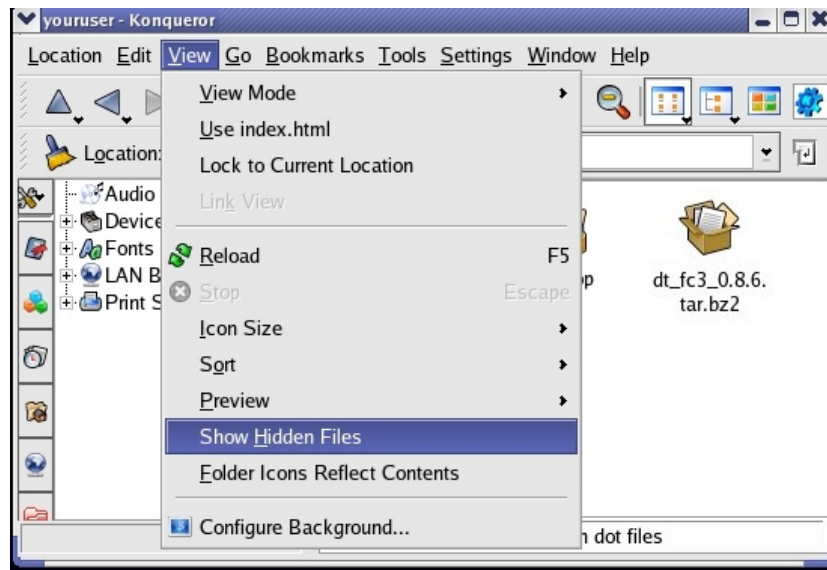


Figure 14. Show Hidden Files.

Below is the command to edit:

```
gedit .bashrc &
```

or double click in the **.bashrc** file to open the **gedit** text editor.

The file will probably be similar to this:

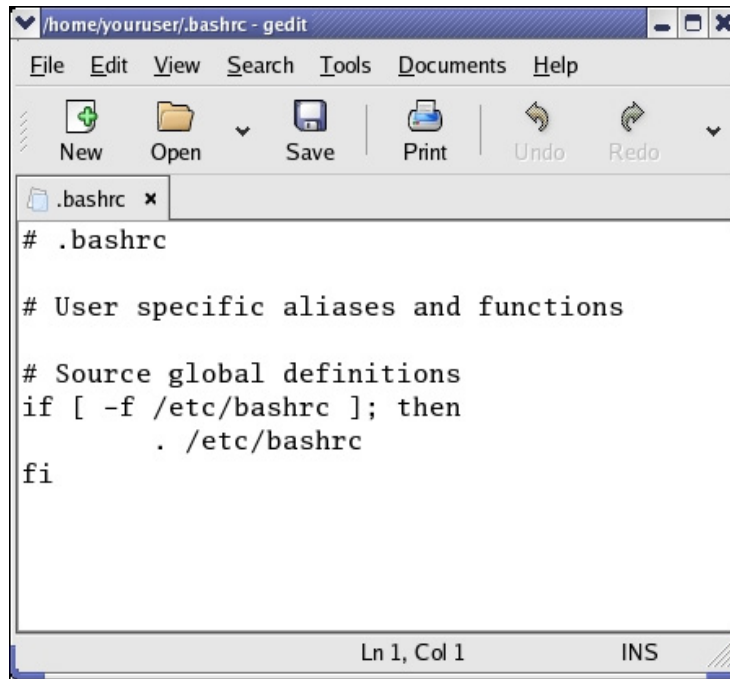
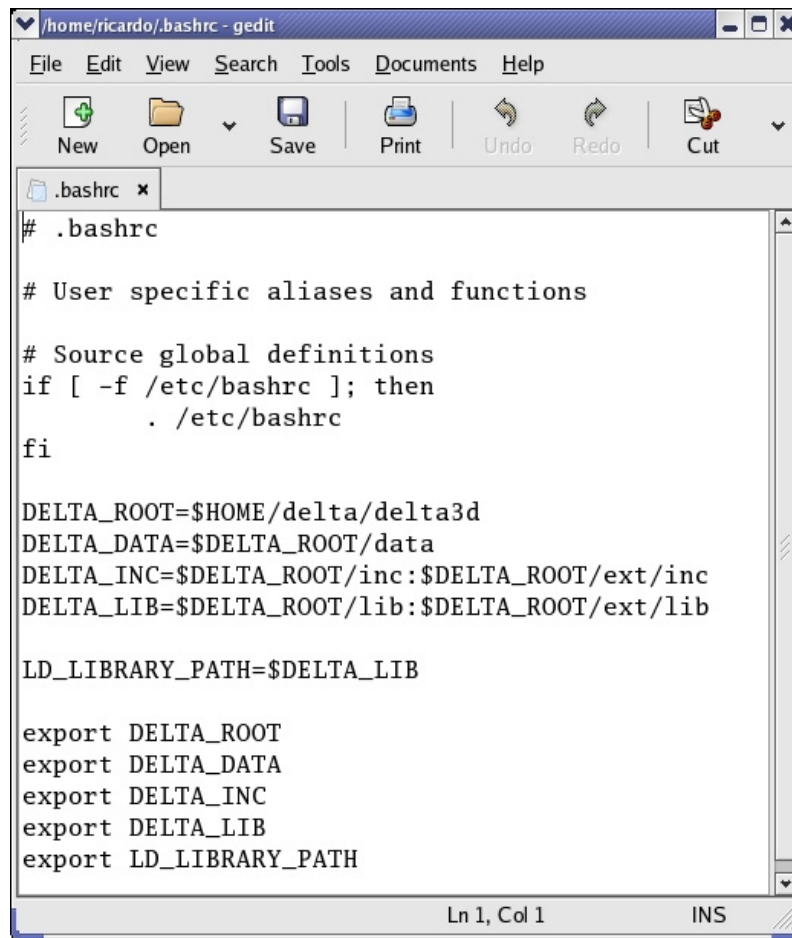


Figure 15. Old .bashrc File.

The following lines must be added to the end of the file:

```
DELTA_ROOT=$HOME/delta3d
DELTA_DATA=$DELTA_ROOT/data
DELTA_INC=$DELTA_ROOT/inc:$DELTA_ROOT/ext/inc
DELTA_LIB=$DELTA_ROOT/lib:$DELTA_ROOT/ext/lib
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$DELTA_LIB
export DELTA_ROOT
export DELTA_DATA
export DELTA_INC
export DELTA_LIB
export LD_LIBRARY_PATH
```


The file will be similar to:

A screenshot of a gedit editor window titled '/home/ricardo/.bashrc - gedit'. The window has a menu bar with 'File', 'Edit', 'View', 'Search', 'Tools', 'Documents', and 'Help'. Below the menu bar is a toolbar with icons for 'New', 'Open', 'Save', 'Print', 'Undo', 'Redo', and 'Cut'. The main text area contains the following code:

```
# .bashrc

# User specific aliases and functions

# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi

DELTA_ROOT=$HOME/delta/delta3d
DELTA_DATA=$DELTA_ROOT/data
DELTA_INC=$DELTA_ROOT/inc:$DELTA_ROOT/ext/inc
DELTA_LIB=$DELTA_ROOT/lib:$DELTA_ROOT/ext/lib

LD_LIBRARY_PATH=$DELTA_LIB

export DELTA_ROOT
export DELTA_DATA
export DELTA_INC
export DELTA_LIB
export LD_LIBRARY_PATH
```

The status bar at the bottom indicates 'Ln 1, Col 1' and 'INS'.

Figure 16. New .bashrc File.

After saving the file all command windows must be closed.

To check if the variables are correctly set, a new command window can be opened and the command **env** typed, checking if the following lines are listed:

```
DELTA_INC=/home/youruser/delta3d/inc:/home/youruser/delta3d/ext/inc
DELTA_ROOT=/home/youruser/delta3d
DELTA_DATA=/home/youruser/delta3d/data
DELTA_LIB=/home/youruser/delta3d/lib:/home/youruser/delta3d/ext/lib
LD_LIBRARY_PATH=:/home/youruser/delta3d/lib:/home/youruser/delta3d/ext/lib
```

INSTALLING SCONS

SCONS (<http://www.scons.org>) is the tool used to compile and build Delta3D files and projects. The **SConstruct** and **SConscript** files presented in Delta3D source code subdirectories are high-level Python scripts that tell SCONS how to build Delta3D.

On the tool website, there is an RPM distribution package available that can be downloaded to **/home/youruser**. To install, it is necessary to assume the root id opening a terminal window (in KDE pressing F4 with the window selected), command:

```
su -
```

After that, the command rpm with some flags to extract and install the files:

```
rpm -ivh scons-0.96.1-1.noarch.rpm
```

(The scons rpm version used as an example was scons-0.96.1-1.noarch.rpm.)

The **SConstruct** and **SConscript** files will do all the compiling and linking work. For Delta3D libraries and examples these files are already created, but for other external projects they need to be created. A template version is presented in the section “Linux Hello World.”

BUILDING DELTA UTILITIES AND EXAMPLES

The process to build Delta3D libraries and put them in the right place is described below.

To compile, in the delta3d root directory (**/home/youruser/delta3d**) just type:

```
scons
```

To install the shared libraries in the lib folder it is necessary to specify a prefix like that:

```
scons install prefix=/home/youruser/delta3d
```

The installation can be confirmed by checking the delta3d lib (/home/youruser/delta3d/lib) folder and looking for the files created.

The Konqueror window will look like this:

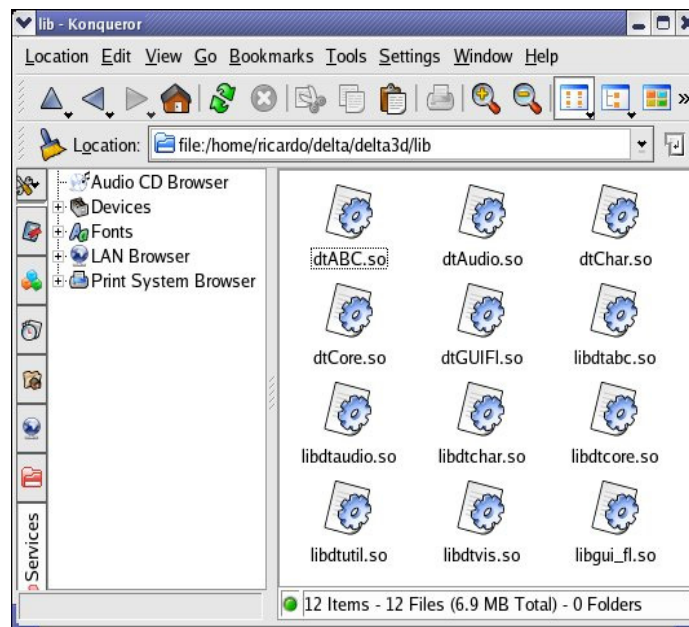


Figure 17. Extract Delta3D Lib Files.

Now the examples and utilities can be built by typing:

```
scons examples
```

```
scons utilities
```

There is a lot of documentation about the scons tool on the scons web site (<http://www.scons.org>). Another option is to ask scons for help:

```
scons -h
```

Some of the options included are:

- Q - Quiet output
- j N - Number of jobs to use, help for multiple processors
- c - Clean out the previous build
- prefix=path - Path to in which to install Delta3D
- mode=debug|release - 'debug' builds with debugging symbols, 'release' builds with optimizations enabled
- noWarnings - Turns off all compiler warnings

If your RTI and Python are already installed, the HLA and Python libraries will be automatically compiled in the first scons command. But they can also be built later. To build HLA-related libraries, examples, and utilities:

```
scons hle
```

To build the Python bindings and example:

```
scons python
```

To test the examples there are two options:

1) Navigating to the examples folder and double clicking in the KDE executable file icon:



2) Opening a terminal window and in the program folder (F4) and typing:

```
./programName
```

Below are some screen shots of two examples running on LINUX:

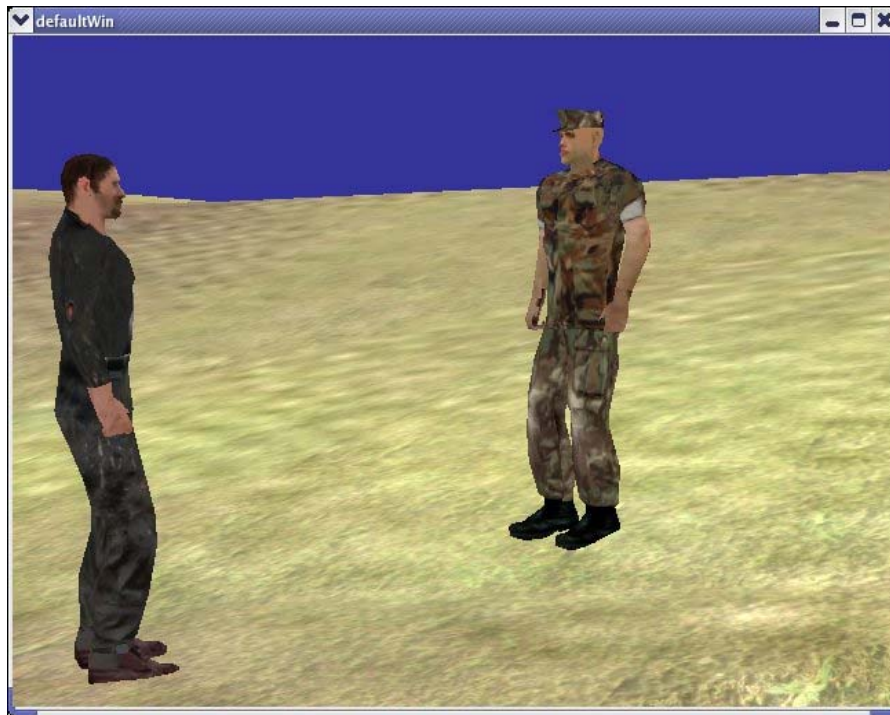


Figure 18. TestCharacter Screenshot.



Figure 19. TestEffects Screenshot.

To run the utilities the same thing can be done. This screenshot is from the Viewer running on Linux with a model loaded:

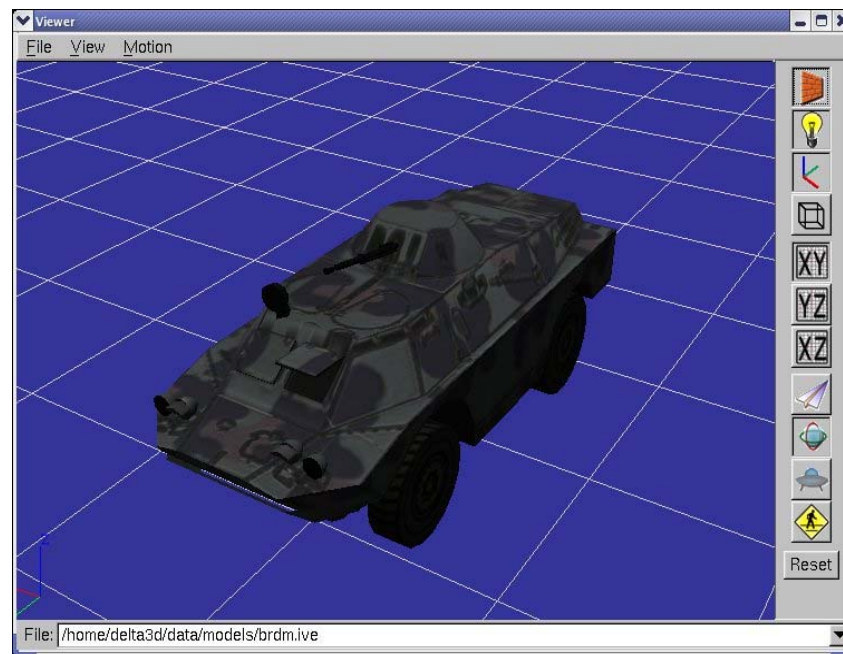


Figure 20. Viewer Screenshot.

ADDING DESKTOP LINKS AND ICONS

To make the Linux workstation easier to use it is interesting to use some desktop links and icons to facilitate access to the programs and folders, as illustrated in the figure below:

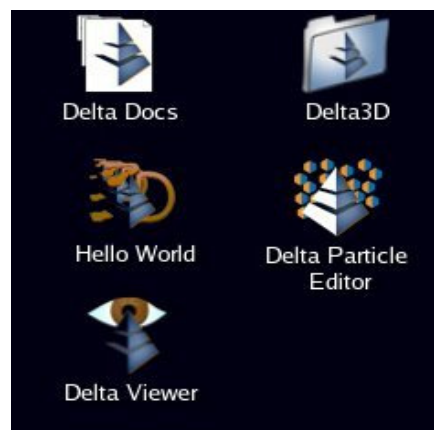


Figure 21. Desktop Icons Screenshot.

To link to the folders Delta Docs and Delta3D, right click the mouse anywhere on the desktop and select Create New, File, **Link to Location** and in File name, navigate to the Delta3d folder and hit OK.

To change the icon, right click in the icon created and double click in the folder icon. To select the Delta3D icons, chose other icons and browse to the folder where the delta icons were downloaded.

To create the docs folder the process is the same, but select **HTML File** (instead of Link to Location).

To create the links to the Viewer and Particle Editor programs the sequence is the same again, but after selecting File, select **Link to Application** (instead of selecting Link to Location).

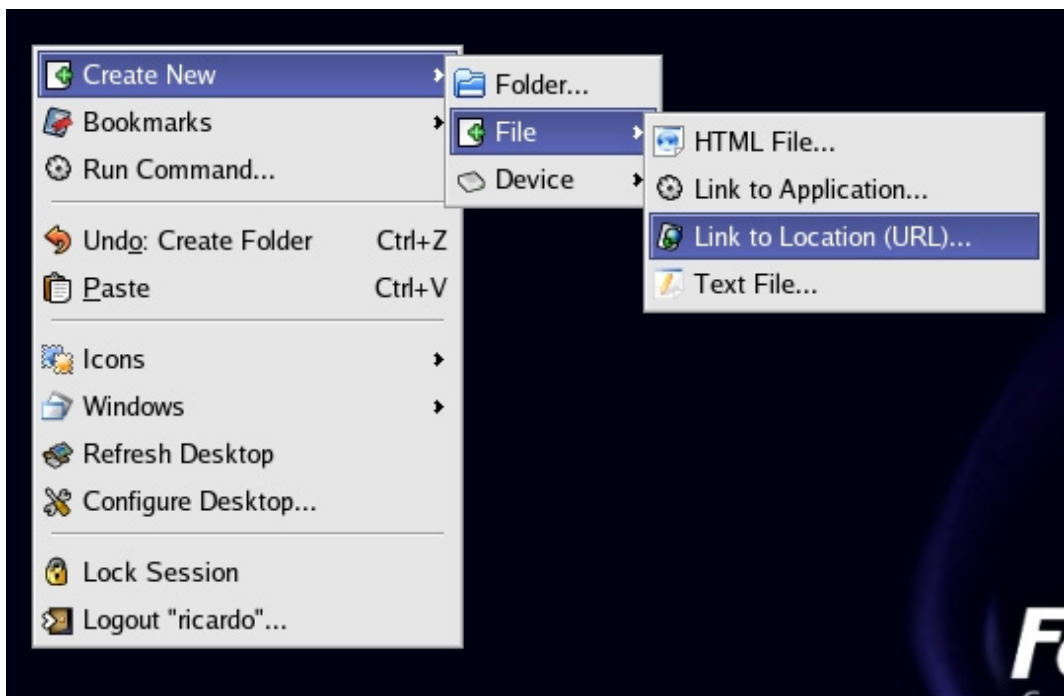


Figure 22. Create New File Desktop.

LINUX HELLO WORLD

Below is the C++ code for a simple Hello World program. It is portable to any other OS supported by Delta3D. This first code is from the **Project1.h** header file:

```

#include "dtCore/dt.h"
#include "dtABC/dtabc.h"

using namespace dtCore;
using namespace dtABC;
using namespace std;

class Project1 : public Application
{
public:
    //Constructor
    Project1(string configFilename):Application(configFilename){

        //setup the data file search paths for the config file
        //and the models files
        SetDataFilePathList("../;../data");

        //Generating the default config file
        //Setting the path
        string path = osgDB::findDataFile(configFilename);
        //verifying if the config file already exists, if not generate
        if (path.empty()) {
            GenerateDefaultConfigFile();
        }

        // Loading a 3D model
        // Instantiate the object related to the model
        Object *obj = new Object("Text");
        // Load the model file, in this case a OpenFlighth model (.flt)
        obj->LoadFile("HelloWorld.flt");
        // Adding the object to the scene as a Drawable object
        GetScene()->AddDrawable(obj);

        // Adjusting the Camera position
        // Instantiating a transform object to
        // store the camera position and attitude
        Transform camPos;

        //coordinates are x      y      z
        camPos.SetLookAt(0.f, -100.f, 20.f, //position
                        0.f, 0.f, 0.f, //look At
                        0.f, 0.f, 1.f); //up Vector
        GetCamera()->SetTransform(&camPos);

        // Setting a motion model for the camera
        OrbitMotionModel *omm = new OrbitMotionModel(GetKeyboard(),GetMouse());
        // Setting the camera as a target for the motion model.
        // The object (the hello world 3D text)
        // will be static at 0,0,0 and the camera
        // will move using the right clicked mouse.
        omm->SetTarget(GetCamera());
    };

    //Default destructor
    ~Project1(){ };
};

```


This second code is from the **Project1.cpp** file:

```
#include "Project1.h"

main()
{
    //instantiate the application and look for the config file
    Project1 *app = new Project1("config.xml");
    // putting a title to explain how end the application
    app->GetWindow()->SetWindowTitle("Hit escape to exit");
    // configuring the application
    app->Config();
    // running the simulation loop
    app->Run();
    return 0;
}
```

To compile the program using Linux, it is necessary to create an **SConstruct** file like the one presented below:

```
import os
import SCons.Util

env = Environment()

# append the outside env to ours
for K in os.environ.keys():
    if K in env['ENV'].keys() and K in ['DELTA_ROOT', 'DELTA_INC', 'DELTA_LIB']:
        env['ENV'][K] = SCons.Util.AppendPath( env['ENV'][K], os.environ[K] )
    else:
        env['ENV'][K] = os.environ[K]

# set compiler options
env.Append( CPPPATH      = ['/usr/X11R6/include' ] +
env['ENV']['DELTA_INC'].split( os.pathsep ) )
env.Append( CXXDEFINES = [ ] ) # for debug mode, add 'DEBUG' and '_DEBUG'
env.Append( CXXFLAGS    = ['-O2', '-pipe', '-Wall' ] ) # for debug mode, add '-g' and replace '-O2' with 'O0'
env.Append( LIBPATH     = ['/usr/X11R6/lib' ] +
env['ENV']['DELTA_LIB'].split( os.pathsep ) )

# source files
project1Srcs = ['Project1.cpp']

# list your lib dependencies here
project1Deps = [ 'OpenThreads', 'osg', 'osgDB', 'osgGL2', 'osgFX',
'osgParticle', 'osgTerrain', 'osgText', 'osgUtil', 'plibsg', 'plibul',
'plibjs', 'Producer', 'gdal', 'fltk', 'Xxf86vm', 'dtcore', 'dtabc', 'dtutil'
]

# build it!
env.Program('Project1', project1Srcs, LIBS = project1Deps )
```

This file can be used as a template for other projects - all that is needed is to change the source file and the dependencies lists for the new project options.

To compile it, type this command in a project folder terminal window:

```
scons
```

To run, double click in the executable icon or use the command line option in a terminal window:

```
./HelloWorld
```

Here is a screenshot of the **HelloWorld** program running on Linux:



Figure 23.

HelloWorld Screenshot.

APPENDIX A REFERENCES (last accessed July 2005)

www.fedorafaq.org

rpm.livna.org

www.fedora.redhat.com

www.scons.org

www.delta3d.org

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. APPLICATION DEVELOPED USING OSS/FS TOOLS

INTRODUCTION

This application was developed to test the many tools and the process proposed in this thesis. It is an example of a PC Naval Visualization Application developed using exclusively open source software. The scenario presented is an Anti-Submarine exercise but it could be easily expanded, with small modifications, to other naval scenarios.

There are references to the source code from three programs from the Delta3D 0.7.0 Open Source Engine Public Beta Test CD distributed during the I/ITSEC 2004. It was developed in C++ and XML using primarily the high level classes from the Delta3D engine.

LICENSE SCHEMA PROPOSED

Following the license schema ideas from the Delta3D Engine, the license schema proposed for this application or any other similar application is to use flexible licenses in the libraries and models and any compatible license in the application.

The engine and libraries can be distributed under flexible open source licenses like LGPL, MIT/X or BSD. The authoring tools under any other license, but generating data under open file formats. The Application developed, the models and data under any different compatible license, even proprietary licenses if allowed.

REFERENCES

The models used are from the references below:

- DDG - Delta3D Asset Library
- Submarine - 3D Café
- Frigate - GCB Open Source Strategic Game

Code references:

From the Delta3D 0.7.0 Public Beta Test CD distributed in the 2004

I/ITSEC:

- Plane Guard
- Driver
- Firefigther

TOOLS, ENGINE AND ENVIRONMENT

During the development process the following open source authoring tools where used:

- For 3D Modeling and Editing - Blender3D
- Textures Creation and Editing - GIMP
- For Audio Recording and Editing - Audacity

The operational system installed in the PC hardware was Linux Fedora Core 3 under the kernel 2.6.12-1.1372_FC3 and the C++ compiler was GCC 3.4.0.

To edit the source code, the EMACS text editor was customized and the SCONS tool was used to automate the process.

The Delta3D Visual Simulation Engine version used was the 8.6 with the OpenSceneGraph library dependency in version 0.9.8. From the engine utilities, the Viewer was used to view the models before selection and file format conversion when needed. The real time Particle System Editor was also used to edit the particle systems to generate the spray and wake of the ships.

HIGH LEVEL CLASS HIERARCHY

This is a high level UML static diagram of the class hierarchy and its relation to the Delta3D classes. The dtCore and dtABC classes are from the Delta3D engine and the pcnviz classes are the ones developed in the application.

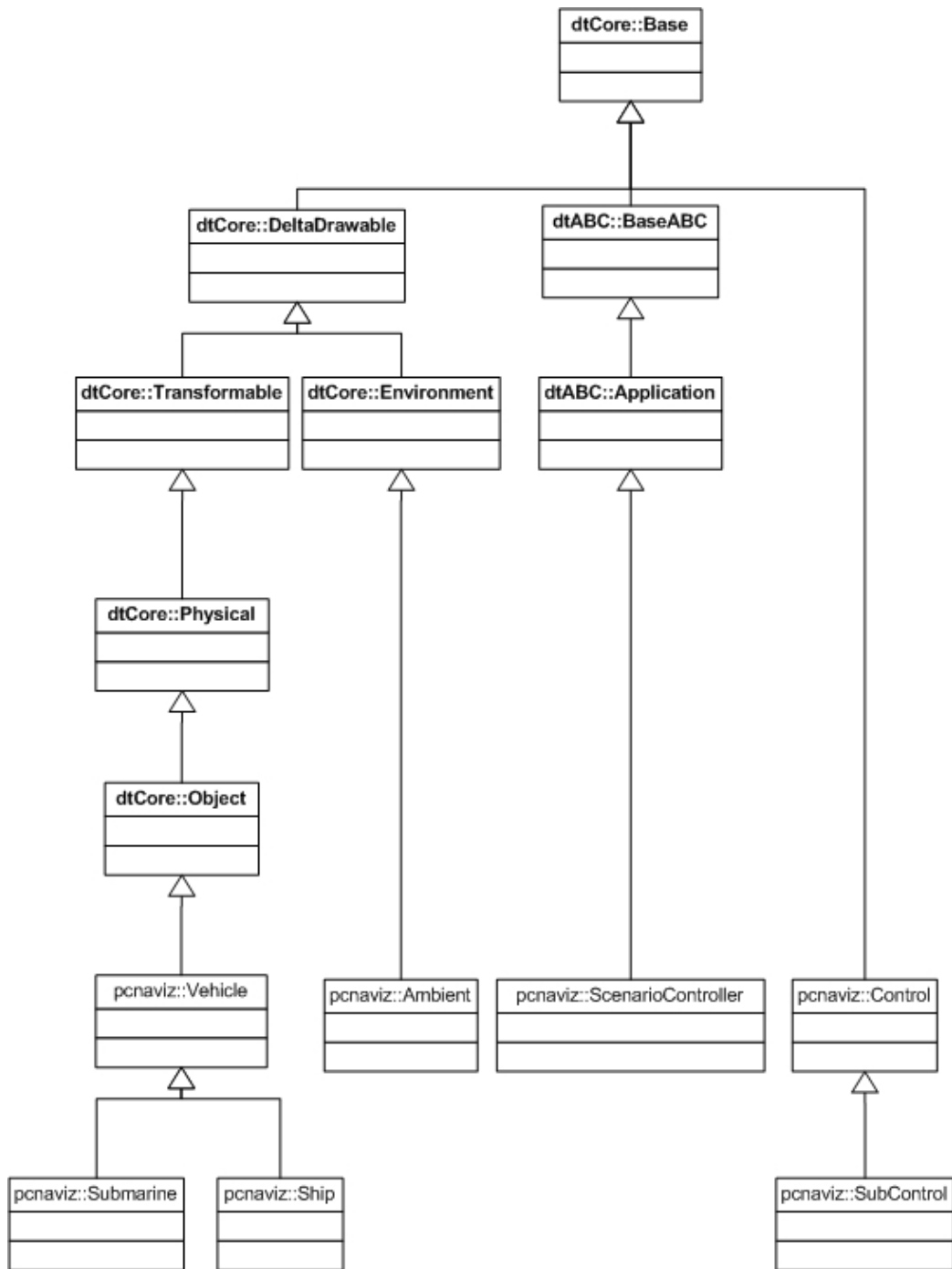


Figure 24. Application Class Hierarchy.

SCREENSHOTS



Figure 25. The DDG Model.

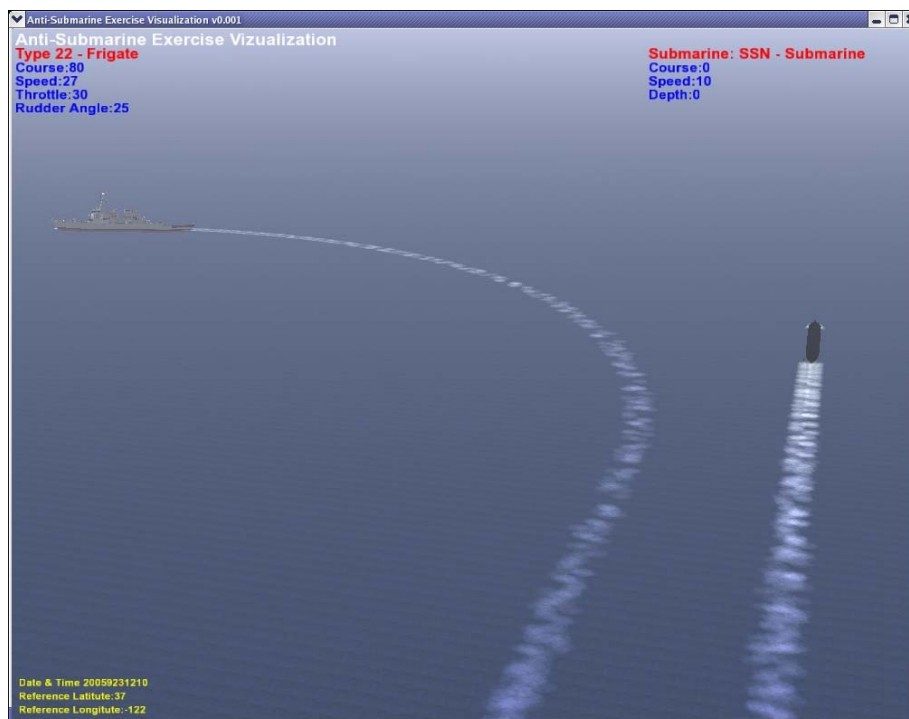


Figure 26. DDG Turn and the Submarine in the Surface.

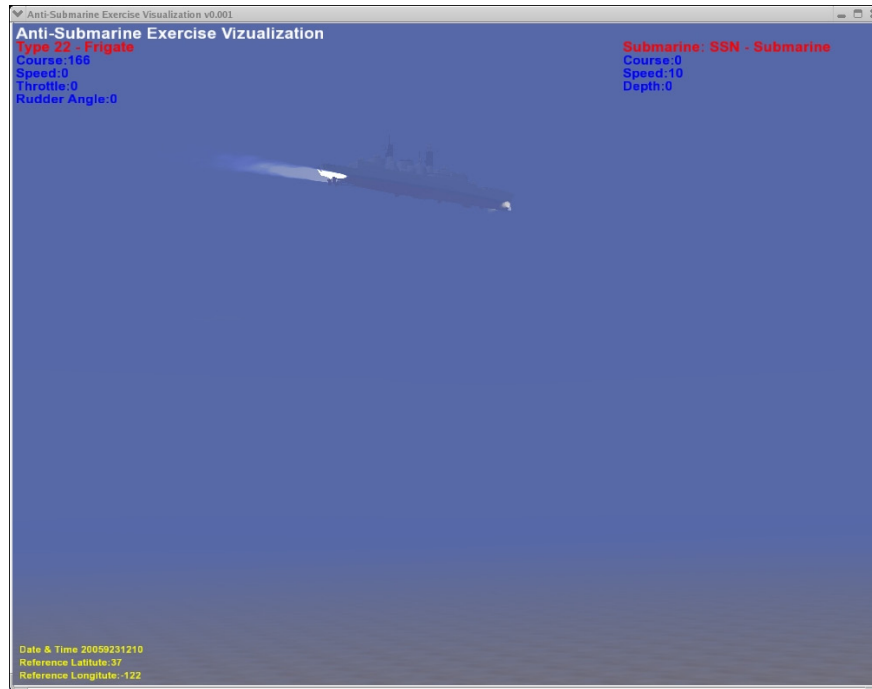


Figure 27. Underwater Camera with a Frigate View.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. Moore, Gordon E., "Cramming More Components Onto Integrated Circuits," *Electronics*, Vol. 38, No. 8, April 19, 1965.
2. "Defense Modeling and Simulation Office," U.S. Department of Defense, Modeling and Simulation Office, (*M&S*) *Glossary DoD 5000.59-M*, <http://www.dmsi.mil/>, (last accessed on September 8, 2004).
3. Caird, J. K., "Persistent Issues in the Application of Virtual Environment Systems to Training," Department of Psychology, University of Calgary (1996).
4. Furness, T. A and Barfield, W., *Virtual Environment and Advanced Interface Design, Chapter 1*, Oxford University Press, New York, 1995.
5. Slater, M., "Measuring Presence: A Response to the Witmer and Singer Presence Questionnaire," *Presence: Teleoperators and Virtual Environments*, 8(5), pp. 560-56, 1999.
6. Heilig, M.L., "El Cine del Futuro: The Cinema of the Future," *Presence: Teleoperators and Virtual Environments*, 1 (3), pp. 279-294, 1992.
7. Psotka, J., Lewis, S.A., and King, D., "Effects of Field of View on Judgments of Self-Location: Distortions in Distance Estimations Even When the Image Geometry Exactly fits the Field of View," *Presence*, Vol. 7, No. 4, 1998.
8. *Handbook of Virtual Environment Technology*, by Lawrence Erlbaum Associates, Inc. (ed: 2002), Chapter 4 Virtual Auditory Displays by Shilling and Shinn-Cunningham.
9. Miner, N. and Caudell, T., "Computational Requirements and Synchronization Issues for Virtual Acoustic Displays," *Presence*, Vol. 7, pp. 396-409, 1998.
10. Prensky, Marc, *Digital Game-Based Learning*, McGraw-Hill, 2001.
11. "Modeling and Simulation – Linking Entertainment and Defense," Computer Science and Telecommunications Board, National Research Council, National Academy Press, Washington D.C., 1997.
12. DARWARS Project, www.darwars.com, (last accessed April 28, 2005).
13. Macedonia, Michael, "Games Simulation and the Military Education Dilemma," EDUCAUSE Information Resources Library, 2001, <http://www.educause.edu/ir/library/pdf/ffpiu018.pdf>.

14. Rose, Frank, "The Lost Boys," *Wired Magazine*, 12-08, August 2004.
15. Prensky, Marc, "Has Growing Up Digital and Extensive Video Game Playing Affected Younger Military Personnel's Skill Sets?" I/ITSEC Conference, 2003, <http://www.marcprensky.com/writing/>.
16. Silberman, Steve, "The War Room," *Wired Magazine*, 12-09, September 2004.
17. Zyda, M., Mayberry, A., McCree, J., Davis, M., "From Viz-Sim to VR to Games: How We Built a Hit Game-based Simulation," *Organizational Simulation: From Modeling & Simulation to Games & Entertainment*, W.B. Rouse and K.R. Boff (Eds.), New York: Wiley, 2005.
18. Erwin, Sandra I., "\$65K Flight Simulator Draws Skepticism from Military Buyers," *National Defense Magazine*, November 2000, [cited 2004 August 18], <http://www.nationaldefensemagazine.org>.
19. Maguire, F., Van Lent, M., Prensky, M., Tarr, R., "Defense Combat Sim Olympics – Methodologies Incorporating the 'Cyber Gaming Culture'," I/ITSEC 2002, <http://www.marcprensky.com/writing/>.
20. Prensky, M., "Why Not Simulation?" <http://www.marcprensky.com/writing/default.asp>, (last accessed September 8, 2004).
21. *Handbook of Virtual Environment Technology*, by Lawrence Erlbaum Associates, Inc. (ed:2002), Chapter 26 Technology Management and User Acceptance of Virtual Environment Technology by David Gross
22. Parent, Rick, *Computer Animation, Algorithms and Techniques*, San Francisco, CA: Morgan Kaufman Publishers, 2002.
23. St. Laurent, Andrew M., *Understanding Open Source and Free Software Licensing*, Sebastopol, CA: O'Reilly, 2004.
24. Open Source Definition, Open Source Initiative Web Site, <http://www.opensource.org>, (last accessed July 2005).
25. GPL License Definition, Free Software Foundation Web Site, <http://www.fsf.org>, (last accessed July 2005).
26. GNU Project Web Site, <http://www.gnu.org/licenses/license-list.html>, (last accessed July 2005).

27. Mozilla Project Web Site, <http://www.mozilla.org/MPL>, (last accessed July 2005).
28. Trolltech Web Site, <http://www.trolltech.com/products/qt/opensource.html>, (last accessed July 2005).
29. Article published at <http://www.serverwatch.com/news/article.php/3497586>, (last accessed July 2005).
30. Wheeler, David A., "How to Evaluate Open Source Software/Free Software Programs," http://www.dwheeler.com/oss_fs_eval.html, (last accessed July 2005).
31. "A Guide to Open Source Software for Australian Government Agencies," <http://www.agimo.gov.au/infrastructure/oss>, April 2005, (last accessed July 2005).
32. Presentation from Dr. Rudolph Darken of the Naval Postgraduate School MOVES Institute Open House, August 2004.
33. Brooks, Frederick P., "The Mythical Man-Month: Essays on Software Engineering," Addison-Wesley.
34. ATI and NVIDIA websites, <http://www.ati.com> and <http://www.nvidia.com>, (last accessed July 2005).
35. Sourceforge and Freshmeat websites, <http://sourceforge.net> and <http://freshmeat.net>, (last accessed July 2005).
36. Google help website, <http://www.google.com/help/features.html#link>, (last accessed July 2005).
37. KDE and GNOME websites, <http://developer.kde.org/documentation/standards/kde/kde-style.html> and <http://developer.gnome.org/projects/gup/hig/2.0/>, (last accessed July 2005).
38. The Mitre Corporation, "Use of Free and Open-Source Software (FOSS) in the U.S. Department of Defense, Version 1.2.04, January 2, 2003," (<http://www.mitre.org/work/sepo/library/SoftwareEngineering/OpenSourceSoftware/dodfoss.pdf>), (last accessed July 2005).
39. OpenGL website, <http://www.opengl.org>, (last accessed July 2005).
40. SGI website, <http://www.sgi.com>, (last accessed July 2005).
41. OpenAL website, <http://www.openal.org>, (last accessed July 2005).

42. Delta3D Visual Simulation Engine website, <http://www.delta3d.org>, (last accessed July 2005).
43. OpenSceneGraph website, <http://www.openscenegraph.org>, (last accessed July 2005).
44. Open Dynamics website, <http://ode.org>, (last accessed July 2005).
45. GNU Image Manipulation Program website, <http://www.gimp.org/>, (last accessed July 2005).
46. Blender 3D Website, <http://www.blender3d.com>, (last accessed July 2005).
47. Audacity website, <http://audacity.sourceforge.net>, (last accessed July 2005).
48. Ogre3D website, <http://www.ogre3d.org>, (last accessed August 2005).
49. Panda 3D website, <http://www.panda3d.org>, (last accessed August 2005).

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California